Simulation and learning in decision processes

Richard Anthony Jones

A thesis submitted to the University of Bristol in accordance with the requirements of the degree of Doctor of Philosophy in the Faculty of Science.

1998

Department of Mathematics

University of Bristol

U. K.

Abstract

In this thesis we address the problem of adaptive control in complex stochastic systems when the system parameters are both known and unknown. The type of models we consider are those which, in the full information case, are known as Markov Decision Processes.

We introduce versions of two new algorithms, the optimiser and the p-learner. The optimiser is a simulation based method for finding optimal values and optimal policies when the system parameters are known. The p-learner is an algorithm for learning about the state transition probabilities; we use it in conjunction with the optimiser when the system parameters are unknown.

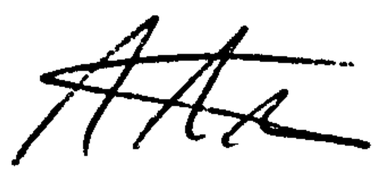
We carefully discuss the choice of different components in the different versions of the algorithms, and we look at two extended case studies to evaluate their performances over a range of different learning parameters. In each case, we compare the results with that of a deterministic method. We also address the convergence of the solutions generated by the optimiser.

Acknowledgements

I extend my thanks to my supervisors, Dr. E. J. Collins and Professor P. J. Green, for their help during this study and EPSRC for financial support. I would also like to take the opportunity to thank Mike Harvey for his proof reading and Tim Downie, Nicky Welton and Rob Menezes for their helpful suggestions. A great big thanks must go to my parents, sister, grandparents and dog Suzi for their love and kindness. I would also like to thank Bernard Silverman, Guy Nason, John McNamara, Dave Green, William Boyd, Richard Lewis, Richard Kerswell, Colm Caulfield, John Walker, Bill Hopkins and John Evans. Finally I would like to thank my friends and the rest of my family for their encouragement.

Author's Declaration

The work described in this thesis was conducted in the Department of Mathematics, University of Bristol and has not been submitted for any other degree or diploma of any examining body. All the material described herein is the original work of the author, except where otherwise acknowledged.



Richard Jones

Contents

Chapter 1

Introduction and Literature Review

1.1 Introduction

In this thesis we consider the problem of learning about and controlling uncertain systems by means of possibly uncertain data. In the past, conventional methods have been developed to find solutions to problems in which system parameters are known. In some applications, however, it is unreasonable to expect these parameters to be known. This is because explicit models of the system are unavailable and the behaviour of the system can only be observed as a whole. Potential applications range from condition monitoring to the operation of industrial robots. Such problems can be formulated as stochastic sequential decision problems.

A simple model frequently used to represent problems such as these is the so-called Markov decision process (Ross 1983). Markov decision processes are general models used in systems evolving in discrete time under Markov assumptions, under the influence of an external action and incurring costs. Costs accumulate over time and the objective is to control the system (using a policy) while minimising the total discounted infinite horizon cost (a value function). In this

thesis we investigate the development and application of stochastic methods for optimisation and adaptive inference, which solve these types of problems.

A commonly used computational technique for finding optimal policies and associated optimal value functions is Dynamic Programming (DP), which is an iterative deterministic approach that can be used when the parameters of the system are known. However, DP is computationally expensive when applied to problems where state and action spaces are large. The computational effort grows exponentially with the dimensionality of the problem. This phenomenon is known as Bellman's "curse of dimensionality" (Bellman 1957). Bertsekas and Tsitsiklis (1989), on the other hand, developed Asynchronous DP where at each iteration computation is only centred around a subset of states, precluding the sort of exhaustive search performed by conventional DP. However, in some applications system parameters are unknown and deterministic algorithms such as standard DP and Asynchronous DP are not directly applicable.

The need to handle adaptive inference for unknown system parameters has motivated the design of stochastic DP learning algorithms called Neuro-Dynamic Programming. Initially it was thought that an explicit model of the system was needed to approximate optimal solutions (termed the "curse of modelling") (Bertsekas and Tsitsiklis 1996). But recent analysis has shown that a simulator can be used instead. Neuro-Dynamic Programming is a simulation based approach which allows the focusing of the computational effort in the state and action spaces to be done in a way that is adaptive to the situation and does not require extensive algebra or analysis to set up. There are two types of algorithms: direct and indirect algorithms (Barto and Singh 1990). Indirect algorithms use a conventional DP algorithm such as Asynchronous DP and if necessary estimate the unknown parameters of the system to find optimal solutions, whereas direct algorithms use an iterative relaxation learning algorithm to estimate parameters that directly specify the control rule. Both approaches learn how to make good decisions via simulation, with the use of built in learning mechanisms. These mechanisms split

into two classes and they balance the exploration and exploitation of state and action spaces (Thurn 1992). One class of method (undirected) chooses states and actions according to a probability distribution, the other (directed) chooses states and actions using heuristics.

In the rest of this chapter we discuss in detail the relevant literature, from Markov Decision Processes through to the more relevant existing Neuro-Dynamic Programming algorithms. In Chapter 2 we describe the methodology we have developed; a simulation based method for finding optimal value function and optimal policies when the system parameters are known. We then extend and successfully apply the idea to cases where the system parameter values of the model are unknown. In Chapters 3 and 4 we show how the methodologies can be applied to simple illustrative case studies. In Chapter 4 we also show how the methodology can be updated in response to the problems that arise. In both case studies we compare the results with that of the deterministic method (DP). Chapter 5 discusses ongoing and directions for future work, and in Chapter 6 we state our conclusions.

1.2 Markov Decision Processes

In this thesis we restrict attention to Markov decision processes considered over an infinite horizon time. Consider a finite state space $S = \{1, \ldots, n\}$. At each discrete time point t, a controller observes the system's current state $i_t \in S$, which evolves probabilistically over time, and selects an action $u_t \in U(i)$ from a finite set of decisions (or controls). If state i is observed and action u is selected, an immediate cost c_t is incurred with expectation $c_i(u)$. The system then moves from state i to state $j \in S$ with probability $p_{ij}(u)$ where,

$$p_{ij}(u) = Pr\{i_{t+1} = j \mid i_t = i, u_t = u\},$$

and,

$$\sum_{j \in S} p_{ij}(u) = 1 \quad \forall \quad i \in S \text{ and } u \in U(i).$$

Let B denote an upper bound on the immediate costs $c_i(u)$. The immediate costs $c_i(u)$ and the state transition probabilities $p_{ij}(u)$ for all $i \in S$ and $u \in U(i)$ are assumed known and fixed at each time t. In this system costs accumulate over time and future costs are discounted at rate α. The objective of the controller is to find a policy minimising the expected value of the total discounted infinite horizon cost, defined in Sections 1.2.1 and 1.2.2 respectively. The controller is therefore confronted with a trade off between choosing actions that produce low immediate costs and choosing actions that move to states with small associated long term costs (see Denardo (1982) for a discussion of this trade off).

1.2.1 Policies

To minimise the expected value of the total discounted cost over time, the controller must find a decision rule that associates an action with each state. A decision rule at time t is defined as $\pi_t(\cdot) = \{\pi_t(1), \ldots, \pi_t(n)\}$. Each decision rule at time t can be thought of as a criterion for choosing a set of actions, one for each state. A policy $\pi(\cdot)$ is a set of decision rules, one for each time point. (Throughout this thesis we will use π and $\pi(\cdot)$ interchangeably). If $\pi_t(\cdot)$ does not change over time it is called a stationary policy, since the policy only depends on the state i_t (Bertsekas and Tsitsiklis 1989). We therefore write $\pi(\cdot)$ for the decision rule (as well as the policy). If $\pi(\cdot)$ minimises the expected value of the total discounted infinite horizon cost for all states $i \in S$, it is called an optimal policy $\pi^*(\cdot)$. Ross (1983) proves that in every finite state finite action infinite horizon discounted problem, there exists at least one optimal policy that is stationary. Therefore in the remainder of this thesis, in each case study we look at, there will be only a single decision rule we need to find.

1.2.2 Value Functions

The expected value of the total discounted infinite horizon cost for each initial state $i \in S$ using policy $\pi(\cdot)$ is defined as,

$$v_\pi(i) = E_\pi \left\{ \sum_{t=0}^{\infty} \alpha^t c_{i_t}(\pi(i_t)) \mid i_0 = i \right\}, \tag{1.2.1}$$

where the parameter $0 < \alpha < 1$ denotes the discount factor and is constant throughout the control problem, and E_π is the expectation assuming policy π. The function defined in Equation (1.2.1) is known as the value function under the policy π. Since $\alpha < 1$, as we move forward in time immediate costs carry more weight than future costs. Furthermore, since $\alpha < 1$ and $|c_i(u)| < B$, the right hand side of Equation (1.2.1) is bounded above, and $|v_\pi(i)| < \frac{B}{1-\alpha}$ (Ross 1983).

Given a policy π the system visits and chooses a state i and an action $\pi(i)$ respectively, and the controller incurs an immediate cost $c_i(\pi(i))$. With probability $p_{ij}(\pi(i))$ the system then moves to the next state of the system j with an associated expected future cost $v_\pi(j)$. As a consequence, the total expected discounted cost of the system for each initial state i using policy π also satisfies the equation,

$$v_\pi(i) \;=\; c_i(\pi(i)) + \alpha \sum_{j \in S} p_{ij}(\pi(i)) v_\pi(j) \quad \forall \quad i \in S. \tag{1.2.2}$$

1.2.3 Optimal Policies and Optimal Value Functions

The optimal value function $v^*(\cdot)$ is defined by the equation,

$$v^*(i) = \inf_\pi \{v_\pi(i)\} \quad \forall \quad i \in S.$$

A policy π^* is said to be optimal if,

$$v_{\pi^*}(i) = v^*(i) \quad \forall \quad i \in S.$$

1.2.4 The Optimality Equation

Ross (1983) shows that the component-wise optimal value function $v^*(i)$ is the unique bounded solution to the optimality (or Bellman) equation,

$$v(i) = \min_{u \in U(i)} \left\{ c_i(u) + \alpha \sum_{j \in S} p_{ij}(u) v(j) \right\} \quad \forall \quad i \in S. \tag{1.2.3}$$

Furthermore, if $\pi(\cdot)$ is a decision rule for which each action $\pi(i)$ minimises the right hand side of the Bellman equation, the stationary policy $\pi(\cdot)$ is optimal.

1.2.5 Greedy Policies

Barto *et al.* (1995) describe a set of successive approximation schemes that solve MDP type problems. However, before they present these algorithms, they further link up the relationship between policies and value functions. In this section we will use their definitions and discuss the concept of policies being greedy with respect to their own value function.

Let $v(\cdot)$ denote a function on S (we can think of v as the value function for some stationary policy).

Define the function $v(\cdot, \cdot)$ by,

$$v(i, u) = c_i(u) + \alpha \sum_{j \in S} p_{ij}(u) v(j) \quad \forall \quad i \in S \text{ and } u \in U(i). \tag{1.2.4}$$

An action $\pi(i)$ is said to be a greedy action with respect to the value function $v(i)$ if,

$$\pi(i) \in \arg\min_{u \in U(i)} \{v(i, u)\} \quad \forall \quad i \in S \text{ and } u \in U(i). \tag{1.2.5}$$

In addition, if all of the actions $\pi(i)$ for all $i \in S$ satisfy Equation (1.2.5) above, then $\pi(\cdot)$ is called a greedy policy with respect to $v(\cdot)$. From Equation (1.2.3), any greedy policy with respect to the optimal value function is an optimal policy.

In the next section we introduce iterative algorithms that generate sequences of value functions and greedy policies over time, in order to calculate $v^*(\cdot)$ and $\pi^*(\cdot)$.

1.3 Dynamic Programming

When all the parameters in the model are known (that is when the state transition probabilities $p_{ij}(u)$ and the immediate costs $c_j(u)$, are assumed known for all states $i, j \in S$ and actions $u \in U(i)$), the stochastic sequential decision problem can be solved using a one-step successive approximation procedure called dynamic programming (DP). DP is an iterative approach to finding an optimal value function $v^*(\cdot)$ and an optimal policy $\pi^*(\cdot)$ for an MDP. The algorithms successively generate a sequence of values functions $v_k(\cdot)$ and policies $\pi_k(\cdot)$ defined for each stage of the computation $k = 0, 1, \ldots$ that converge to $v^*(\cdot)$ and $\pi^*(\cdot)$ as $k \to \infty$. We will discuss two variants of the DP algorithm called value iteration; the Pre-Jacobi and the Gauss-Seidel method. An alternative method to value iteration is called policy iteration, but we do not actually use this method in this thesis.

1.3.1 Value Iteration

The original idea of value iteration dates back to Bellman (1957). In conventional value iteration (full) DP , at each stage $k = 0, 1, \ldots$ of the iteration we evaluate the value function (the total cost) for all the possible state-action $(i \in S, u \in U(i))$ pairs as part of the updating procedure. Then for each state $i \in S$ we choose the action that minimises the value function for that particular state. The set of actions that minimise the value function for all the states $i \in S$ at stage k forms an optimal policy at stage k.

1.3.1.1 Pre-Jacobi DP

Blackwell (1965) proposed an algorithm known as the Pre-Jacobi (PJ) DP method in operations research literature, or alternatively the Synchronous DP method in the artificial intelligence literature. In this algorithm, where in full DP the total cost is updated for all the possible state action pairs at each stage k, the total

cost consists of updates from the last stage for each possible successor state $j \in S$. The value function $v_k(i)$ at stage k and state $i \in S$ is an approximation to the optimal value function $v^*(i)$ at stage k of the iteration. It is updated by sweeping through all of the states $j \in S$ before the local update at state i is complete. Thus for each state $i \in S$ and stages $k = 0, 1, \ldots$ let,

$$v_{k+1}(i,u) \;=\; c_i(u) + \alpha \sum_{j=1}^{n} p_{ij}(u) v_k(j) \quad \forall \quad i \in S \quad \text{and} \quad u \in U(i), \tag{1.3.6}$$

then,

$$v_{k+1}(i) \;=\; \min_{u \in U(i)} \{v_{k+1}(i,u)\} \quad \forall \quad i \in S, \tag{1.3.7}$$

and,

$$\pi_{k+1}(i) \;\in\; \arg\min_{u \in U(i)} \{v_{k+1}(i,u)\} \quad \forall \quad i \in S. \tag{1.3.8}$$

The value function $v_{k+1}(\cdot)$ consists of all the local updates $v_{k+1}(i)$ which are computed simultaneously for all states $i \in S$ using the value functions from the previous stage at every new update.

The terminal costs $v_0(i)$ are assumed known at the start for each state $i \in S$. In an infinite horizon discounted problem they can be set to any value. However, the bigger the difference between $v_0(i)$ and $v^*(i)$ for all $i \in S$, the longer the algorithm will take to converge (Bertsekas and Tsitsiklis 1989). In this dissertation we assume that all the immediate costs $c_i(u) \geq 0$ for all $i \in S$ and $u \in U(i)$. If we assume that the terminal cost $v_0(i) \leq v^*(i)$ for each state $i \in S$ and $c_i(u) \geq 0$, then for each stage k, $v_k(i) \leq v^*(i)$ for all $i \in S$. It is a well known fact that $v_k(\cdot)$ tends to $v^*(\cdot)$ as $k \to \infty$. Denardo (1982) proves it by first obtaining the result,

$$\sup_{i \in S} |v_{k+1}(i) - v^*(i)| \leq \alpha \sup_{i \in S} |v_k(i) - v^*(i)| \quad \forall \quad i \in S. \tag{1.3.9}$$

Equation (1.3.9) is a contraction mapping around the unique fixed point $v^*(i)$ for all states $i \in S$. This ensures that $v_k(i)$ converges to $v^*(i)$ as $k \to \infty$ and the convergence rate has an upper bound (Jaakkola, Jordan, and Singh 1994). Denardo

(1982), on the other hand, proves the convergence by writing Equation (1.3.9) in the form,

$$\begin{aligned} \| v_{k+1} - v^* \| &\leq \alpha \| v_k - v^* \|, \\ &\leq \alpha^2 \| v_{k-1} - v^* \|, \\ &\leq \alpha^{k+1} \| v_0 - v^* \|, \end{aligned} \tag{1.3.10}$$

where $\|v_k - v^*\| = \sup_{i \in S} |v_k(i) - v^*(i)|$. Therefore as $k \to \infty$, $\alpha^{k+1} \to 0$, which implies $v_k(\cdot)$ tends to $v^*(\cdot)$ as $k \to \infty$. A similar proof of convergence is presented in Puterman (1994).

Furthermore, if there are n states and m is the largest number of actions in any one state, then each iteration requires an order of $O(mn^2)$ operations (Barto, Bradtke, and Singh 1995).

1.3.1.2 Gauss Seidel

Porteus (1975) proposed the Gauss-Seidel DP method. The total cost of each state-action pair is updated slightly differently from that of the Pre-Jacobi method. In the Pre-Jacobi method we updated the current value function at each state $i \in S$ and stage k simultaneously, by keeping all the other value functions on the RHS of Equation (1.3.6) fixed. The Gauss-Seidel method, on the other hand, uses the most recent information available to it. At each stage k the value function at each state $i \in S$ is updated in numerical order from state 1 through to n. The value function at state $i \in S$ is computed using the latest value function estimates from states 1 to $i-1$ in the present stage, and states i to n in the previous stage.

Let $v_k(i)$ be the estimate of the optimal value function $v^*(i)$ at stage k. Thus for each state $i \in S$ and stages $k = 0, 1, \ldots$, the sequence of value functions and policies are generated by letting,

$$v_{k+1}(i,u) = c_i(u) + \alpha \sum_{j=1}^{i-1} p_{ij}(u) v_{k+1}(j)$$

$$+ \quad \alpha \sum_{j=i}^{n} p_{ij}(u) v_k(j) \quad \forall \quad i \in S \quad \text{and} \quad u \in U(i), \qquad (1.3.11)$$

then,

$$v_{k+1}(i) \quad = \quad \min_{u \in U(i)} v_{k+1}(i,u) \quad \forall \quad i \in S, \qquad (1.3.12)$$

and,

$$\pi_{k+1}(i) \quad \in \quad \arg\min_{u \in U(i)} \{v_{k+1}(i,u)\} \quad \forall \quad i \in S, \qquad (1.3.13)$$

where $\pi(i)$ is the action that minimises $v_{k+1}(i,u)$. Also, $\pi_{k+1}(\cdot)$ is the policy that minimises $v_{k+1}(\cdot,\cdot)$. As in the Pre-Jacobi method, the terminal costs $v_0(i)$ for each state $i \in S$ are assumed known at the start of each simulation. The convergence proof of this algorithm can be found in Bertsekas and Tsitsiklis (1989).

1.4 Asynchronous DP (ADP)

Asynchronous DP (ADP) was originally proposed by Bertsekas (1982), and then later revised and presented by Bertsekas and Tsitsiklis (1989). The algorithm is used in many of the stochastic algorithms we present in the remainder of this chapter, as it was originally designed for multi-processor systems which had communication delays and no common clock (Barto, Bradtke, and Singh 1995). However, the Pre-Jacobi and Gauss Seidel methods are special cases. In contrast to the algorithms described above, the current value function and policy estimate at each stage k are updated at only a subset of states $S_k \subseteq S$, while their values for the other states remain unchanged. Thus, for each stage $k = 0, 1, \ldots$ we let,

$$v_{k+1}(i,u) = \begin{cases} c_i(u) + \alpha \sum_{j \in S} p_{ij}(u) v_k(j) & \text{if } i \in S_k \text{ and,} \\ v_k(i,u) & \text{if } i \notin S_k, \end{cases} \qquad (1.4.14)$$

then,

$$v_{k+1}(i) \quad = \quad \min_{u \in U(i)} \{v_{k+1}(i,u)\}, \qquad (1.4.15)$$

and,

$$\pi_{k+1}(i) \in \arg\min_{u \in U(i)} \{v_{k+1}(i,u)\}. \tag{1.4.16}$$

The terminal costs $v_0(i)$ for each state $i \in S$ are specified before the start of any simulation. Furthermore, if the immediate costs $c_i(u)$ and the true state transition probabilities are known for each $i, j \in S$ and $u \in U(i)$, and each state is visited on an infinite number of occasions as $k \to \infty$, then the solutions $v_{k+1}(\cdot)$ and $\pi_{k+1}(\cdot)$ are guaranteed to converge to $v^*(\cdot)$ and $\pi^*(\cdot)$ respectively as $k \to \infty$ (see Bertsekas and Tsitsiklis 1996). Note if $S_k = S$ for each k then the algorithm reduces to the Pre-Jacobi method, whereas if $S_0 = \{1\}, S_1 = \{2\}, \ldots, S_{n-1} = \{n\}, S_n = \{1\}, S_{n+1} = \{2\}$ etc. the algorithm acts like the Gauss-Seidel method.

1.5 Iterative Relaxation Schemes (IRS)

Before we discuss Neuro-Dynamic Programming algorithms we describe a stochastic algorithm that estimate $v^*(\cdot)$ and $\pi^*(\cdot)$ called Iterative Relaxation Schemes (IRS). IRS algorithms are a set of algorithms that are crucial to the convergence of Neuro-Dynamic Programming algorithms. Iterative Relaxation Schemes stochastically generate sequences $v_{k+1}(\cdot)$ and $\pi_{k+1}(\cdot)$ for each stage k where DP and ADP algorithms are special cases (Singh 1994). They are of the form,

$$\begin{aligned} \textbf{new update} &= \textbf{old update} + \textbf{learning rate} \\ &\times \textbf{(new estimate - old update)}. \end{aligned}$$

More formally let T be a mapping that produces a new estimate $T(v_k)$ of the solution $v^*(i,u)$ for the state-action pair (i,u), using the approximate value functions at the previous stage; then the updated solutions for each stage k are defined by setting,

$$v_{k+1}(i,u) = v_k(i,u) + \gamma_k(i,u)\{T(v_k(i,u)) - v_k(i,u)\} \quad \forall \quad i \in S \quad \text{and} \quad u \in U(i), \tag{1.5.17}$$

and,

$$v_{k+1}(i) = \min_{u \in U(i)} \{v_{k+1}(i,u)\} \quad \forall \quad i \in S, \tag{1.5.18}$$

and,

$$\pi_{k+1}(i) \in \arg\min_{u \in U(i)} \{v_{k+1}(i,u)\} \quad \forall \quad i \in S. \tag{1.5.19}$$

The learning rate $\gamma_k(i,u)$ defines how much weight goes into the old update v_k and the new estimate $T(v_k)$. An example of the mapping T can be shown in Asynchronous DP where $T(v_k(i,u)) = v_{k+1}(i,u)$, defined in Equation (1.4.14) above. Therefore in ADP, $\gamma_k(i,u)$ equals one for all state-action pairs ($i \in S, u \in U(i)$) and stages k. In this case it can be shown that the solutions of $v_{k+1}(\cdot)$ and $\pi_{k+1}(\cdot)$ will converge to $v^*(\cdot)$ and $\pi^*(\cdot)$ respectively, as long as each state-action pair is visited on an infinite number of occasions as $k \to \infty$ (Kaelbling, Littman, and Moore 1996). However, in some of the methodology we review later in this chapter, the solutions will only converge to their corresponding optimal values if the learning rate parameter decreases slowly for each state action pair. An example of this is shown in the Q-Learning method described in Section 1.6.3.1 below.

1.6 Direct and Indirect Learning Algorithms

1.6.1 Introduction

In this section we review a dichotomy of Neuro-Dynamic Programming algorithms developed for solving stochastic optimisation problems when at most partial information about the system is available. The different types of algorithms are called *Indirect* and *Direct* methods of learning. Indirect methods (model based) use a form of Asynchronous DP to estimate $v^*(\cdot)$ and $\pi^*(\cdot)$, and estimate unknown parameters of the model such as the immediate costs $c_i(u)$ and the true

state transition probabilities $p_{ij}(u)$ for all $i, j \in S$ and $u \in U(i)$ depending on model assumptions. These estimated system parameters are then used to define a control rule. Direct methods (model free), on the other hand, estimate $v^*(\cdot)$ and $\pi^*(\cdot)$ directly without learning about the explicit models of the system. In general, direct methods require less memory and less computation per control action (Barto and Singh 1990). We first describe some indirect methods and then direct methods. Note in some of the algorithms we describe below we assume the system is moving forward in time, therefore index k is substituted for index t.

1.6.2 Indirect Methods

1.6.2.1 Simulation Based Value Iteration

A stochastic variant of straight forward full DP is a method called Simulation Based Value Iteration (Bertsekas and Tsitsiklis 1996). As in full DP, all of the system parameters such as the immediate costs $c_i(u)$ and the state transition probabilities $p_{ij}(u)$ for all $i \in S$ and $u \in U(i)$ are fully specified. The algorithm was designed for learning about the optimal value function $v^*(\cdot)$ and the optimal policy $\pi^*(\cdot)$ when computation in very large state spaces become infeasible. Assuming an initial value function $v_0(\cdot)$, a stochastically chosen state i_k and a discount factor $0 < \alpha < 1$, the method computes a sequence of value functions $v_k(\cdot)$ and policies $\pi_k(\cdot)$ using intermediate functions $v_k(\cdot, \cdot)$ and the following equations,

$$v_{k+1}(i,u) = \begin{cases} c_i(u) + \alpha \sum_{j \in S} p_{ij}(u) v_k(j) & \text{if } i = i_k, \\ v_k(i,u) & \text{if } i \neq i_k, \end{cases} \tag{1.6.20}$$

$$v_{k+1}(i) = \min_{u \in U(i)} \{v_{k+1}(i,u)\}, \tag{1.6.21}$$

$$\pi_{k+1}(i) \in \arg\min_{u \in U(i)} \{v_{k+1}(i,u)\}. \tag{1.6.22}$$

Bertsekas and Tsitsiklis (1996) show that if each state $i \in S$ is visited often enough, then $v_{k+1}(\cdot) \to v^*(\cdot)$ and $\pi_{k+1}(\cdot) \to \pi^*(\cdot)$. However, time constraints and

large state spaces will prevent calculations being iterated to convergence due to a lack of exploration of the state space. We present different schemes for exploring state and action spaces in Section 2.3.2. In addition, Bertsekas and Tsitsiklis (1996) point out the algorithm takes longer to converge if the initial estimates of $v^*(\cdot)$ are far from the final solution. Also, the order in which the current value function estimate and policy estimate is updated at each particular state, influences the algorithm's rate of convergence.

The algorithm is a special of case of ADP, but with different ramifications. It is also similar to the algorithm we developed called the *optimiser*, presented in Section 2.2 which was developed independently before Bertsekas and Tsitsiklis's (1996) was published. It is also an example of Barto et al.'s (1995) real-time DP where the controller only receives information about one state at each stage k. Our algorithm, on the other hand, is presented slightly differently. The motivation behind our algorithm was to concurrently run it with the *p-learner*, the probability learner (see Section 2.3), when the state transition probabilities are unknown.

1.6.2.2 Adaptive Real Time Dynamic Programming

Adaptive Real Time Dynamic Programming (ARTDP) was developed by Barto *et al.* (1995), where the state transition probabilities and (possibly) the immediate costs are estimated by recording state transitions and past actions. These parameters are then used in the appropriate equations to estimate $v^*(\cdot)$ and $\pi^*(\cdot)$. The algorithm is a three step iteration: (1) update the model, (2) update the current value function and policy estimates, and (3) take actions. Step (1) is a statistical estimation problem where the true state transition probabilities and (possibly) immediate costs are estimated via maximum likelihood techniques. For instance, for each time t let the current model of the system consist of the maximum likelihood estimates, $\hat{p}_{ij}^t(u)$, of the true state transition probabilities for all

states $i, j \in S$ and actions $u \in U(i)$. Let $x_{ij}^u(t)$ be the observed transitions from state i to state j using action u before time t. Let $x_i^u(t) = \sum_{j \in S} x_{ij}^u(t)$ be the number of times the system was in state i and employed action u before step time t. The maximum likelihood estimates of the true state transition probabilities for all $i, j \in S$ and $u \in U(i)$ are therefore,

$$\hat{p}_{ij}^t(u) = \{x_{ij}^u(t)\} \left\{ \sum_{j \in S} x_{ij}^u(t) \right\}^{-1}. \tag{1.6.23}$$

A similar formula is used for estimating costs if the immediate costs are unknown. Step (2) is a Simulation Based Value Iteration technique except that the algorithm is indexed by time t rather than stages k. Here the most recent changes of the model are used in the DP calculation, for example Equation (1.6.23) would be substituted for the true state transition probabilities in Equation (1.6.20). Thus the DP calculation uses Bertsekas's (1987) *certainty equivalence principal* whereby current value function and policy estimates would be optimal if the current model was correct. Finally Step (3) uses the Boltzmann exploration technique to choose actions. The Boltzmann exploration method is used to explore randomised polices, and as $t \to \infty$ the method focuses on the greedy actions to find an improved policy (see Section 1.7.2.2).

Certainty equivalent methods make efficient use of the data generated, but the method often requires a great deal of experience to achieve good results. Using a random exploration function such as the Boltzmann Exploration sometimes proves to be costly. If the system prematurely focuses on the greedy actions the current value function and policy estimates may converge to suboptimal solutions and never recover (Kaelbling, Littman, and Moore 1996). Sato *et al.* (1988) estimate their true state transition probabilities in exactly the same way as Equation (1.6.23), but their algorithm is based around policy iteration, not value iteration. They also use a different scheme for choosing actions which we discuss in Section 1.7.4.1. Sato *et al.* (1988) prove that their method's solutions converge to the optimal if good learning parameters are chosen, or approximately

optimal if they are not. Fiechter (1994), on the other hand, estimates both the state transition probabilities and immediate costs in his learning algorithm using the same maximum likelihood techniques as stated above. However, even though the algorithm is based on value iteration, it is a trial-based method. In his algorithm the system is run over a sequences of time called trials. Each trial is reset after N iterations and starts with the original initial state i_t at time 0. Fiechter (1994) proves that the solutions generated converge to approximate optimal solutions with high probability.

The algorithms we present in Chapter 2 are extensions to these ideas. In Step (1) we use Bayesian rather maximum likelihood techniques, as using prior information about the system improves the initial conditions. We execute Steps (2) and (3) using two separate algorithms, running in parallel. We extend the ideas of the Boltzmann Exploration Method in Section 2.2.2. Also, in Section 4.13 we present methodology which resets the system but at every step time t, not every N iterations. In addition, we do not reset each trial with the initial state i_0. The motivation behind the development of our reset operation was based on the empirical study described in Chapter 4.

1.6.3 Direct Methods

1.6.3.1 Q-Learning

Watkins (1989) proposed a reinforcement learning algorithm called *Q-Learning*, as an extension of Sutton (1988) (see Section 1.6.3.2 below). Q-Learning does not use an explicit model of the system for estimating $v^*(\cdot)$ and $\pi^*(\cdot)$. Instead, system parameters are used explicitly in an entirely non-parametric way. Barto *et al.* (1995) classes Q-Learning as a direct approach to learning, since the method estimates $v^*(\cdot)$ and $\pi^*(\cdot)$ but without estimating the system parameters. The method belongs to the family of Iterative Relaxation Algorithms illustrated in Section 1.5 above.

1.6 Direct and Indirect Learning Algorithms

For any policy π and any state-action $(i \in S, u \in U(i))$ pair we can define a Q-value as,

$$Q_\pi(i,u) = c_i(u) + \alpha \sum_{j \in S} p_{ij}(u) v_\pi(j), \tag{1.6.24}$$

where,

$$v_\pi(i) = c_i(\pi(i)) + \alpha \sum_{j \in S} p_{ij}(\pi(i)) v_\pi(j).$$

Let us suppose that we observe the current state of the real system $i_t = i$ and we choose an action $u_t = u$ (different schemes for choosing actions can be found in Section 1.7). Given the current state-action pair (i,u), if the next state of the system $i_{t+1} = j$ is chosen with probability $p_{ij}(u)$, the sample $v_\pi(j)$ is an unbiased estimate of the total quantity $\sum_{j=1}^{n} p_{ij}(u) v_\pi(j)$. Furthermore c_t is an unbiased estimate of $c_i(u)$. Therefore an estimate of Equation (1.6.24) is given by,

$$c_t + \alpha v_\pi(j).$$

Since Q-Learning is a special case of IRS (see Equation (1.5.17)), assuming a current state-action pair (i_t, u_t) and functions $Q_t(\cdot,\cdot)$ and $v_t(\cdot)$, Q-values and policies at each time $t = 0, 1, \ldots$ can be approximated by,

$$Q_{t+1}(i,u) = \begin{cases} \{1 - \gamma_t(i,u)\} Q_t(i,u) & \\ \qquad + \gamma_t(i,u)\{c_t + \alpha v_t(j)\} & \text{if } i = i_t,\ u = u_t, \\ Q_t(i,u) & \text{otherwise,} \end{cases}$$

and,

$$\pi_{t+1}(i) \in \arg\min_{u \in U(i)} \{Q_{t+1}(i,u)\}.$$

The learning parameter $0 \leq \gamma_t(i,u) \leq 1$, for each (i,u), $i \in S$ and $u \in U(i)$, starts off at the value 1 and slowly tends to 0 as $t \to \infty$, and are assumed to satisfy the conditions $\sum_{t=1}^{\infty} \gamma_t(i,u) = \infty$ and $\sum_{t=1}^{\infty} (\gamma_t(i,u))^2 < \infty$. Humphrys

(1996), for example, used the learning parameter for updating the pair (i, u) at time t equal to,

$$\gamma_t(i,u) \;=\; (n(i,u))^{-1},$$

where $n(i,u)$ is the number of times action u has been selected in state i prior to time t; this produced some good results. The learning parameter determines how much weight goes in to the previous update $Q_t(i,u)$ and the new estimate $c_t + \alpha v_t(j)$, since the new update $Q_{t+1}(i,u)$ is a combination of the two. Thus when $\gamma_t(i,u) = 1$ the weighting is entirely on the new estimate $c_t + \alpha v_t(j)$, and as $\gamma_t(i,u) \to 0$ no learning is done at all.

In contrast to Barto et al.'s (1995) Adaptive Real Time Dynamic Programming, Q-Learning is not as computationally expensive. For example, suppose that there are n states and that the largest number of actions in any one state is m. Since adaptive RTDP stores all the state transition probabilities and immediate costs in a look up table, the number of storage locations used is of the order $O(mn^2)$. Q-Learning, on the other hand, requires of the order $O(mn)$ storage locations for the current state-action value function estimates. Q-Learning is not as exploration sensitive either, as the method concentrates on the value function for each state-action pair not just each state. Since the Q-values are updated according to the greedy action at the following state, the method can choose whatever action it wishes after that state. As long as each state-action pair is visited often enough and the learning parameter for each state action pair satisfies the criterion stated above, $Q_t(i,u) \to Q^*(i,u)$ as $t \to \infty$ with probability 1. Authors such as Watkins (1989) and Watkins and Dayan (1992) have proved this result. Other authors have proved the result for different variants of the method such as Jaakkola *et al.* (1994), who realised that Q-Learning is connected (but not directly) to stochastic approximation theory (Robbins and Monro 1951). Tsitsiklis (1994), on the other hand, also proved its convergence

using stochastic approximation theory, but the main proof is based on the supermartingale theorem found in Bertsekas and Tsitsiklis (1996).

Peng and Williams (1994) extended the idea of the one-step Q-Learning method, by developing a routine, combining both the Q-Learning and actor/critic approaches to learning. The main advantages are: it can do many jobs at once, it corrects mistakes, and it can transmit data to where it is needed most.

1.6.3.2 Temporal Differences

Sutton (1984) proposed a model free method called Temporal Difference Method $TD(\lambda)$. This laid the foundations for Watkin's Q-Learning method (above). Temporal Differences assigns more credit to states that have been visited more recently and the frequency in which they have occurred. The method does this using an exponential weight $0 \leq \lambda \leq 1$, for example, observations that were made k steps in the past are weighted according to λ^k. Every time a state is observed a trace is initiated, and this marks the state eligible for learning. The updating rule is based on the IRS rule given in Equation (1.5.17) in conjunction with an eligibility function $e(l)$ below for all $l \in S$. Thus,

$$v_{k+1}(i) = v_k(i) + \gamma_k(i)\left\{c_t + \alpha v_k(j) - v_k(i)\right\} e(i), \tag{1.6.25}$$

where,

$$e(i) = \sum_{m=0}^{t} (\lambda\alpha)^{t-m} \delta_{i,i_m}, \tag{1.6.26}$$

$$\delta_{i,i_m} = \begin{cases} 1 & \text{if } i = i_t, \\ 0 & \text{otherwise.} \end{cases} \tag{1.6.27}$$

However, unlike in Q-Learning the updating is applied to every state $l \in S$ according to their eligibility trace function $e(l)$, rather than the immediately previous state i. The trace given in Equation (1.6.27) is known as the conventional trace,

but other traces have been developed (see Singh and Sutton 1996). When $\lambda = 0$ the method is equivalent to Q-Learning and the method is only interested in its most recent observation, whereas when $\lambda = 1$ all states are weighted equally. Tradeoffs between different values of λ can found in Bertsekas and Tsitsiklis (1996). The Temporal Difference method using a general λ is computationally expensive to execute, but requires considerably less computation time for its solutions to converge to their optimal when λ is large (Kaelbling, Littman, and Moore 1996). Many authors have proved the convergence of the method's solutions to their corresponding optimal values, such as Dayan and Sejnowski (1994), Peng and Williams (1994) and Dayan (1992). They all realised that TD(0) is the same as Q-Learning. Jaakkola *et al.* (1994) and Bertsekas and Tsitsiklis (1996), on the other hand, proved TD(λ)'s convergence through its connection with stochastic approximation theory.

1.6.3.3 Dyna Architectures

Sutton's (1990) Dyna architecture combines learning, planning and reactive execution of actions when a complete and accurate model of the system is unknown. The design is based on the theory of DP and it links together the theory of other stochastic iterative learning algorithms, to find approximations of $v^*(\cdot)$ and $\pi^*(\cdot)$, reviewed in the remainder of this chapter. The architecture is also known as Incremental Dynamic Programming (IDP) (Sutton 1991b). Planning is based on the idea of making decision rules. At each time t the controller observes a state, and an action is planned to be taken. It is only when planning is complete, in response to a state, that an action is actually taken. This takes time and an action is only executed fairly quickly when planning is both short and weak. Reactive systems, on the other hand, were developed so that actions could be chosen without delay. These actions are defined as a function of the current state. Dyna combines these two conjectures by using learning algorithms to approximate the

method of DP.

Dyna is based on hypothetical and real experiences (Sutton 1991a). Hypothetical (H) experiences are based on the planning process, whereas real (R) experiences are based on learning about an optimal reactive policy and predicting a new state $j \in S$ given a state $i \in S$ and an action $u \in U(i)$ are fed in to a "black box" (trial and error learning). The generic algorithm is defined as follows:

Repeat indefinitely,

Step (1) (R) Observe the current state $i \in S$ and reactively choose an action $u \in U(i)$,

Step (2) (R) Observe the immediate cost $c_i(u)$ and a new state $j \in S$,

Step (3) (R) Update $v(i,u)$ based on the iterative relaxation scheme below, and update the new policy based on it,

Step (4) (R) Choose a new action $u \in U(j)$ using an exploration strategy,

Step (5) (H) Choose Y state-action pairs (i_Y, u_Y) at random. Update $v(i_Y, u_Y)$ and the new policy based on this hypothetical experience, using the same method as in Step (3).

The exploration strategies used in Dyna are not strictly part of the architecture, because they are only used to mimic what actions the system would take in real life. Dyna is therefore not bound to any particular exploration strategy. Any method can be used, but the capabilities of the different strategies will have a marked effect on the quality and efficiency of the model learnt.

One of the advantages of Dyna is that it is totally incremental and it works well with stochastic systems. Another is that any model free or model based updating scheme can be used in conjunction with Dyna. Therefore, depending on the updating rule Dyna can be used as an indirect or direct method. Barto

and Singh (1990) state that one way of retaining many of the advantages of both indirect and direct methods for learning is to use Dyna. Sutton (1990) successfully uses policy iteration and Q-Learning in conjunction with Dyna called Dyna-PI and Dyna-Q respectively, whereas Kaelbling *et al.* (1996) successfully use Dyna in conjunction with value iteration (Dyna-VI). However, if Dyna-VI is used then the model should be updated between Step (4) and (5) in the algorithm above. The disadvantage of Dyna is that IDP planning requires large amounts of memory. For instance traditional planning methods are based on constructing sample paths and the current value function estimate and policy estimates are updated according to the current state or state-action pair, whereas Dyna updates the current value function estimate policy at Y randomly chosen state-action each time. Kaelbling *et al.* (1996) show that Dyna-VI requires about Y times the computation of Q-Learning at each time step, but requires an order of magnitude fewer steps to converge to an optimal policy.

1.7 Learning Methods for Exploration and Exploitation

1.7.1 Exploration and Exploitation

This section describes the various learning methods developed for choosing states and actions in the direct and indirect learning algorithms described in Section 1.6 above. These methods help the learning algorithms sample state and action spaces. We assume there are limits on the amount of computational time available, thus each algorithm cannot exhaustively sample all state and action pairs. To ensure the current value function estimates and current policy estimates converge to $v^*(\cdot)$ and $\pi^*(\cdot)$ respectively, the algorithms must concentrate their efforts on the most important states and actions. By this we mean the states the system would visit and the actions it would take when it is being controlled by an

optimal policy. To find these important states and actions, the algorithms have to balance learning on the one hand and the desire to focus on what is important on the other. As Thurn (1992) puts it, there are two opposing objectives: 1) sampling state and action spaces to gain knowledge of the system and 2) the knowledge gained must be used to minimise the cost of learning. Point 1) is used to learn about the system (exploration) and 2) uses this information to focus on what is important (exploitation). They are collectively termed the balance between identification and control.

It is difficult to assess what the optimal amount of learning might be as opposed to exploitation. Two of the questions we want to ask are what is the optimal amount of learning and how can we cut the costs of this learning? Too much exploration wastes a great deal of time visiting irrelevant parts of the spaces, but without exploring we cannot minimise the overall costs of the system. They must be used hand in hand.

A taxonomy of learning techniques has been developed, some of which work better in certain situations. An efficient learning technique for one particular problem may be inferior in another. It is apparent in the literature that not many exploration techniques have been developed for choosing states, only actions. The most popular strategy for choosing actions is the Undirected Exploration technique. Undirected Exploration techniques choose states and actions at random according to a given probability distribution. The two most commonly Undirected Exploration techniques for choosing actions, used in finite state and action spaces, are called Semi-Uniform Distributed Exploration and Boltzmann Exploration. We extend these ideas in Section 2.3.2. The Directed Exploration techniques, on the other hand, use exploration rules which are entirely heuristic. Directed Exploration techniques choose states or actions to indirectly move to states that have been visited least often, less recently or perhaps even states that have a high prediction error as compared to the rest. Thurn (1992) states that

the heuristic methods are more efficient, in terms of learning costs and time minimised, than the undirected methods. He has also developed a theorem asserting their superiority over undirected methods for most finite deterministic problems. In this section we first describe Undirected Exploration techniques for choosing actions and then directed Exploration techniques for choosing actions. We end the section by looking at existing methods for choosing states.

1.7.2 Choosing Actions - Undirected Exploration Methods

1.7.2.1 Semi-Uniform Exploration Method

John (1995) considers the Semi Uniform Distributed Exploration Method which is also known as the Random Walk Exporation Method. Let $u^*(i)$ be the current action estimate in state $i \in S$. When in state $i_t = i$ at time t, action $u_t = u$ is chosen with probability,

$$\begin{cases} \{1-p\}|U(i)|^{-1} & \text{if } u \neq u^*(i) \text{ and,} \\ p + \{1-p\}|U(i)|^{-1} & \text{if } u = u^*(i), \end{cases} \tag{1.7.28}$$

where a current action estimate is chosen with a (large) probability p, and an action is chosen uniformly with a (small) probability $1-p$ where p is fixed.

There are both advantages and disadvantages with this method. We address the issues when we compare it with our extension to this method found in Section 2.3.2.1. One of the main disadvantages of this method is that the controller has to choose an apriori global fixed probability p, regardless of the state $i \in S$, before the start of each simulation. This means that the stochastic learning algorithms will learn more about the costs of some states than others. Also, given a fixed probability p, finding the optimal tradeoff between exploration and exploitation becomes virtually impossible.

1.7.2.2 Boltzmann Exploration Method

Numerous authors such as Barto *et al.* (1995) consider the Boltzmann Exploration Method. In the Boltzmann Exploration Method we define a utility function $w_t(i,u)$ for the current state $i_t = i$ and all possible actions $u_t = u \in U(i)$ at time t based on information gained by the system or in most cases the current value function estimates themselves. The method uses these estimates of the utility to discriminate between actions. Initially the magnitudes of the utility functions for state i over all actions $u \in U(i)$ are equal, then as the simulation proceeds the algorithm gains more information and the utility functions start to differ until eventually the greedy action estimate in state i will be chosen with the highest probability.

The method uses a temperature schedule (parameter) which is used to balance exploration and exploitation. It is used in a similar way to that of simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983): as time passes the temperature is reduced so that the algorithm focuses on taking the best actions so far in the simulation rather than exploring and selecting randomised actions. The probability of taking action $u_t = u$ in state $i_t = i$ is,

$$\frac{\exp\left\{Z_t^{-1} w_t(i,u)\right\}}{\sum_{u \in U(i)} \exp\left\{Z_t^{-1} w_t(i,u)\right\}} \quad \text{for all} \quad u \in U(i), \tag{1.7.29}$$

where Z_t is the temperature at time t which starts off at infinity and eventually decreases to zero. Thus when Z_t is large the algorithm chooses actions $u \in U(i)$ in state i with equal probability and when Z_t is small the current action estimate for state i has the highest probability of being selected because it has the lowest utility.

We discuss the pros and cons of this method when we look at the extensions of this method found in Sections 2.3.2.2 and 2.3.2.3. The root problem in this method is that the temperature schedule must be carefully tuned before the method works well. On the other hand, if the overall costs over all actions $u \in U(i)$ for all $i \in S$ are quite far apart, the method will produce good results.

1.7.3 Choosing States - Undirected Exploration Methods

There are no explicit undirected exploration methods for choosing states to update the current value function and policy estimate, other than either moving to states uniformly at random as in Dyna (Sutton 1990) or according to the true state transition probabilities (Bertsekas and Tsitsiklis 1996). The trouble with moving to states uniformly at random is that it is computationally expensive, since the current value function and action estimates at certain states may have already converged to their corresponding optimal values. The problem with moving to states according to the true state transition probabilities is that if the system is unknown, these probabilities may not be available. We address these problems and extend the ideas in Sections 2.2.2 and 4.7.

1.7.4 Choosing Actions - Directed Exploration Methods

1.7.4.1 Counter Based Methods

Thurn (1992) describes a counter based method for choosing actions. At each time t the method counts the number of visits $c(i)$ for each state $i \in S$. The method uses this information to choose actions. This in turn helps the algorithm indirectly visit less explored states. The algorithm selects an action $u_t = u$ in state $i_t = i$ if the function,

$$\{c(i_t)\} \left\{ E[\ c(i'_{t+1}) \ \mid \ i_t = i, u_t = u\ ] \right\}^{-1}, \tag{1.7.30}$$

is maximised, where $E[\cdot|\cdot]$ is the expected value and i'_{t+1} is the possible successor state of state i given action u is chosen. An extension of this method is called *counter based exploration with decay.* At each time t the method counts the number of visits $c(i)$ for each state $i \in S$, and the function $c(\cdot) = c(1), \ldots, c(n)$ is multiplied by a decay factor $0 < \lambda < 1$. The decay factor improves the efficiency of exploration as the most recent visits to particular states are weighted more heavily.

Kaelbling (1993) introduces a similar counter based method called the interval estimation algorithm. The method uses utility functions similar to these described for the Boltzmann Exploration Method described in Section 1.7.2.2. Initially, the method is based on overestimates of a utility function either for each state or state-action pair. An overestimation tells the algorithm that particular actions are being ignored. Each utility is updated in accordance to the number of visits to each state or state-action pair. The algorithm uses the utility function to calculate confidence intervals for the success probabilities of actions being chosen. An action is chosen if its upper bound is highest. However, since eventually these utilities tend to their corresponding true values, so exploration is reduced. Thus a secondary exploration method must be used so that greedy actions may be focused on for the rest of the run. Kaelbling *et al.* (1996) state that the method works well in empirical trials.

Sato *et al.* (1988) designed a counter based method for choosing actions. In this method sufficient exploration is allowed so that the estimated state transition probabilities converge to the true ones. The method records the number of times each action $u \in U(i)$ is not executed in state $i \in S$, from step time 0 to the present step time t. Actions are taken on the basis of maximising an exploratory function, and the method is biased towards choosing greedy actions over those which have not been performed for a while. Barto and Singh (1990) show that this method performs quite well on a simple stochastic adaptive problem.

Sutton (1990), on the other hand, developed a method that keeps records on the number of times each action is selected in each state. The method is called *exploration bonus in Dyna.* If an action is ignored for too long, the success probability of it being chosen is forced to be more favourable. This method is quite popular in many reinforcement learning algorithms.

1.7.4.2 Other Exploration Methods

Puterman (1994) developed an error based exploration method used to eliminate suboptimal actions. This method is based on the semi-norm span. Error based exploration methods are explained in more detail below.

1.7.5 Choosing States - Directed Exploration Methods

1.7.5.1 Error Based Exploration

Error based exploration methods predict errors in the current value function estimate over time. These predictions force the controller to visit regions of the state space where the estimated error is relatively large, and thus makes current solutions to a problem converge faster to their corresponding optimal values (Thurn 1992). Kaelbling *et al.* (1996) describe a method called *Prioritized Sweeping* for stochastic value iteration learning algorithms. Prioritized Sweeping concentrates on updating the current value function estimate where it is needed most. The original paper was presented by Moore and Atkeson (1993). Peng and Williams (1993) present a similar idea based on Q-Learning.

Each state is ranked according to values called *priorities*. The state with the highest priority is sampled to update the current value function and policy estimate. Furthermore, under any action $u \in U(i)$ each state $i \in S$ recalls its *predecessors* - the set of states that have a non zero transition probability to it under any action. If a state is sampled, information is passed back to its predecessors and their priorities are updated. The algorithm is a four step iterative and is defined as follows: Firstly let the current state $i_t = i$ at time and set the priorities for each state to zero then, (1) compute $v_{t+1}(\cdot)$, (2) set the priority for state i to zero, (3) compute the change in the current value function estimate $\Delta = |v_t(i) - v_{t+1}(i)|$, and (4) use Δ to update the priorities of i. For example, if we used Real Time Dynamic Programming, the precedessors'

priorities would be updated to $\Delta\, \hat{p}_{ij}^{t}(u)$ unless their existing priorities exceed that value.

Prioritized Sweeping is much more effective than other sampling methods. Kaelbling *et al.* (1996) show that in a simple problem, an optimal policy was reached (using RTDP with prioritized sweeping) in approximately half the time of Dyna-VI and it took 20 times fewer steps than Q-Learning. However, they do mention that it took twice the computation of Q-Learning.

The Index Method we present in Section 4.7 is similar to prioritized sweeping above, but our method is an undirected rather than a directed method. Our motivation behind the development of this method is also slightly different. We developed the Index Method so that we could look at the case structured problems, rather than looking at the general context. Our ideas came from the numerical evaluation of our original method used in the empirical study described in Chapter 4. Our idea of using indices to update states came from the paper by Gittins (1989), who used indices for choosing optimal actions at each step in sequential bandit problems.

Chapter 2

The Methodology: The Optimiser and P-Learner

2.1 Introduction

The following chapter describes the methodology we have developed to minimise and control stochastic cost problems when system parameters are both known and unknown. We refer to the methods described below when trying to solve the various problems present in the case studies discussed in Chapters 3 and 4. The methodology consists of two algorithms known as the *optimiser* and the *p-learner* which together enable us to execute adaptive inference and control (Jones and Collins 1998).

The optimiser is an algorithm that can be used to solve a problem when both the immediate costs and the state transition probabilities are known. The algorithm represents a new way of solving stochastic cost problems (the old ways are discussed in Section 1.3). The method is a special case of Asynchronous DP. Instead of updating the total cost function in each state $i \in S$ at each iteration as in full DP, the updating is only performed on a subset of states which varies over time.

2.2 The Optimiser

In many problems system parameters such as the state transition probabilities are unknown and the optimiser cannot solve the problem on its own. A separate algorithm called the p-learner (the probability learner) is introduced in conjunction with the optimiser. The p-learner learns about the true state transition probabilities over time by observing transitions made in the past and by taking action. Assuming we have a finite computer budget of size T, we cannot afford to learn too much general information about the overall system. Therefore at first the p-learner spreads its effort uniformly over the whole state and action space, while successively focusing in on the most important state and actions. At each iteration the p-learner gives the current state transition probability estimates to the optimiser and the optimiser uses these estimates in place of the true values in the DP equations.

As well as describing the algorithms above we introduce novel ways of choosing both states and actions in the optimiser and actions in the p-learner. The learning algorithms for choosing actions are extensions of the work presented in John (1995). We illustrate how the algorithms are implemented. We end the chapter by discussing possible realistic applications.

2.2 The Optimiser

2.2.1 The Algorithm

In this section, we describe the optimiser, the algorithm that we shall use in the case studies later in this thesis for approximating the optimal value function $v^*(\cdot)$ and an optimal policy $\pi^*(\cdot)$. We assume for the moment that we have a fully-specified MDP, so that the cost functions $c_i(u)$ and state transition probabilities $p_{ij}(u)$ are assumed known for all $i, j \in S$ and $u \in U(i)$. The full DP value iteration solution for the problem for each stage $k \geq 0$, assuming a discount factor α and

the value function $v_0(\cdot)$ is therefore (see Section 1.3),

$$v_{k+1}(i,u) = c_i(u) + \alpha \sum_{j \in S} p_{ij}(u) v_k(j) \quad \text{for each } u \in U(i), \tag{2.2.1}$$

$$v_{k+1}(i) = \min_{u \in U(i)} \{v_{k+1}(i,u)\}, \tag{2.2.2}$$

$$\pi_{k+1}(i) \in \arg\min_{u \in U(i)} \{v_{k+1}(i,u)\}, \tag{2.2.3}$$

for all $i \in S$ and under standard conditions $v_{k+1}(\cdot)$ and $\pi_{k+1}(\cdot)$ converge to $v^*(\cdot)$ and $\pi^*(\cdot)$ respectively, as k tends to infinity.

Informally, the idea of the optimiser is to update $v_k(i,u)$ using Equation (2.2.1) for only a subset of states i and included actions $u \in U(i)$, at each stage k. The subset of states, denoted S_k, will be chosen randomly, in a manner correlated with a simulation of the process itself. This choice is discussed in Section 2.2.2 below. Our method is therefore a special case of what Bertsekas and Tsitsiklis (1996) describe as Simulation-Based Value Iteration DP.

Our algorithm for obtaining approximations to $v^*(\cdot)$ and $\pi^*(\cdot)$ for each stage $k \geq 0$ assuming the current value function estimate $\widehat{v}_0(\cdot)$ is therefore,

$$\widehat{v}_{k+1}(i,u) = \begin{cases} c_i(u) + \alpha \sum_{j \in S} p_{ij}(u) \widehat{v}_k(j) & \text{if } i \in S_k, \\ \widehat{v}_k(i,u) & \text{if } i \notin S_k, \end{cases} \tag{2.2.4}$$

together with, unchanged from Equations (2.2.2) and (2.2.3):

$$\widehat{v}_{k+1}(i) = \min_{u \in U(i)} \{\widehat{v}_{k+1}(i,u)\}, \tag{2.2.5}$$

$$\widehat{\pi}_{k+1}(i) \in \arg\min_{u \in U(i)} \{\widehat{v}_{k+1}(i,u)\}, \tag{2.2.6}$$

although notice that this minimisation need only be performed for $i \in S_k$: the other values are unchanged.

In applications, it may be unreasonable to assume that the state transition probabilities $p_{ij}(u)$ are known, therefore in Section 2.3 below we discuss the algorithms for estimating $p_{ij}(u)$ in parallel with optimisation.

For a given state $i \in S$ we call $\widehat{v}_{k+1}(i)$ the current value function estimate and $\widehat{\pi}_{k+1}(i)$ the current action estimate for each stage k. In Chapter 5 we show

that the current value function estimate $\widehat{v}_{k+1}(\cdot)$ and the current policy estimate $\widehat{\pi}_{k+1}(\cdot)$ defined above for stage k converge to the true optimal value function $v^*(\cdot)$ and a true optimal policy $\pi^*(\cdot)$ respectively as $k \to \infty$.. This result holds as long as each state $i \in S$ is visited often enough. However, time constraints and large state spaces will prevent calculations being iterated to convergence due to a lack of exploration of the state space (Bertsekas and Tsitsiklis 1996). As the general design of algorithm of Bertsekas and Tsitsiklis (1996) is similar to ours, the same comments apply. In addition, they point out the algorithm takes longer to converge if the initial estimates of $v^*(\cdot)$ are far from the final solution.

One of the criteria we will look at in Chapters 3 and 4 is how closely the current optimal value function and the current optimal policy match the true optimal value function and true optimal policy respectively at each decision epoch. In the remaining sections of this chapter we will look at what happens if we do not know the exact values of the system parameters and discuss how we do the optimisation without them? But before we go on to look at these questions we will discuss one particular new method for choosing subsets of states in the optimiser.

2.2.2 Choosing States

In this section we describe a method we shall use in Chapters 3 and 4 for choosing subsets of states in the optimiser. Throughout this thesis the subset of states S_k we choose at each stage k in the optimiser is of the simplest form $S_k = \{i_k\}$ where i_k is a single state generated at stage k. The general design of algorithm of Bertsekas and Tsitsiklis (1996) also has $S_k = \{i_k\}$, but in their algorithm, given a current state $i_k = i$, the controller chooses a control action $u_k = u \in U(i)$, and the next state of the system $i_{k+1} = j$ is chosen with probability $p_{ij}(u)$. This is where our algorithm differs.

In our algorithm a state $i_k = i$ is chosen to update $\widehat{v}_{k+1}(i)$ at stage k and the

controller then finds its corresponding current action estimate $\widehat{\pi}_{k+1}(i) \in U(i)$, but in addition chooses another action $u_k = u \in U(i)$. This action u_k combined with state i_k determines the next state $i_{k+1} = j$, as described below using Equations (2.2.7) and (2.2.8). The different schemes for choosing actions u_k within each state i_k are discussed in Section 2.3.2. However, it should be stated here that as k increases the actions u_k which the controller eventually focuses on are the current action estimates $\widehat{\pi}_{k+1}(i)$ corresponding to state i_k.

The optimiser generates states as follows: the optimiser assumes an initial state i_0 which is drawn randomly from S. Subsequently, a random sequence of states i_k at stages $k = 1, 2, \ldots, K$ is generated according to a probability distribution defined for each action u,

$$q_{ij}(u) \;=\; p_{ij}^{1/T_k}(u) \left\{ \sum_{l \in S} p_{il}^{1/T_k}(u) \right\}^{-1}, \tag{2.2.7}$$

for all states $i, j, l \in S$ and actions $u \in U(i)$, where K is the total number of stages in the computation, i is the current state, j is the proposed state and l is a possible successor state. The function T_k is the temperature schedule at stage k and is defined at $k = 1, 2, \ldots, K$ as follows:-

$$T_k \;=\; \{1 + \exp(4 - 8k/K)\}. \tag{2.2.8}$$

The function $1/T_k$ used to power up the true state transition probabilities in Equation (2.2.7) is illustrated in Figure 2.1 below for $K = 80000$. The formal definition of $q_{ij}(u)$ is the probability that we make the transition from state i to state j given we use action u currently. Note when $T_k = 1$, $q_{ij}(u) = p_{ij}(u)$ as used by Bertsekas and Tsitsiklis (1996).

The idea of the introducing the temperature schedule so that the optimiser generates the next state according to the probability distribution $q_{ij}(u)$ instead of $p_{ij}(u)$ is to help the optimiser sample all of the states long enough for the current value function estimates and the current action estimates to converge to $v^*(i)$ and $\pi^*(i)$ in each state $i \in S$ respectively. At first because $1/T_k$ is small

2.2 The Optimiser

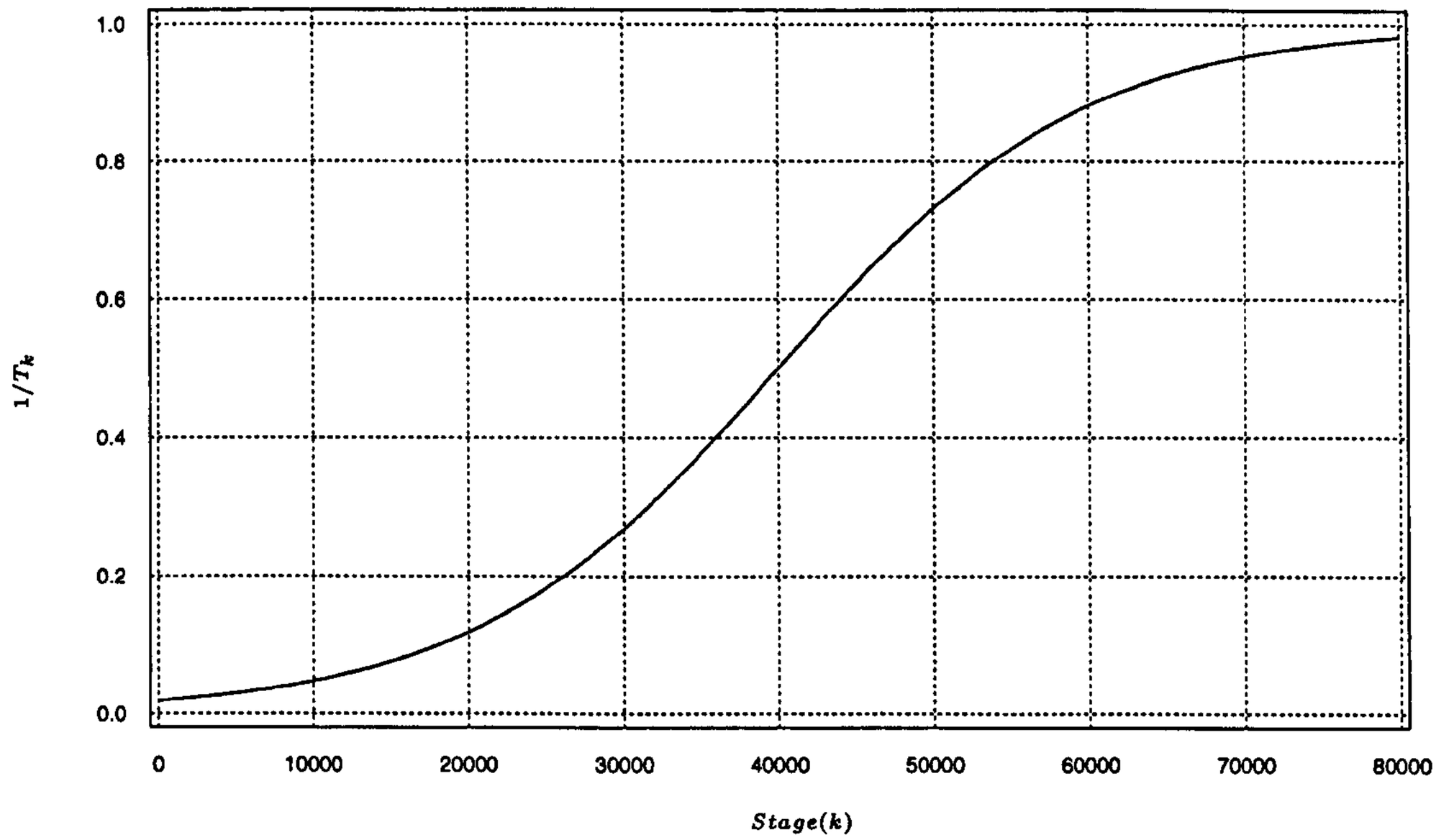


Figure 2.1: The temperature schedule $1/T_k$ is used to power-up and re-normalise the true state transition probabilities as shown in Equation (2.2.7).In the above diagram $1/T_k$ is plotted against stage k where $K = 80000$.

the optimiser samples the whole of the state space with equal probability. Then after $k = K/2$, the optimiser switches to a different regime and tries to sample states according to the true state transition probabilities.

One of the advantages of choosing a dependent S_k approximately following the simulation of the process, over both a random S_k drawn independently from a distribution or a deterministic S_k, is that we do not need to know so much about the process. This will be advantageous in problems where the true state transitions probabilities are unknown. We have already mentioned that as k increases the controller eventually focuses on the current action estimates in each state, therefore the process itself is being controlled to be approximately optimal. Thus the optimiser will concentrate on some states more than others and as a result the computational effort will work harder in these states which is probably the right place for the optimiser to be working. However, following the idea that

we are trying to get the optimiser to work in the states where it really matters, by decreasing the temperature in the way that we do, we allow the process to run itself so that the optimiser will find out which these states are.

2.2.3 Related Problems

The optimiser described above solves infinite discounted stochastic optimisation decision problems. The optimiser can also be tailored to solve optimal stopping problems as we show in Chapter 4.

2.3 The P-Learner

2.3.1 The Algorithm

In this section, we describe the p-learner (the probability learner), the algorithm that we will use in case studies found in Chapters 3 and 4, for estimating the true state transition probabilities. We estimate the true state transition probabilities $p_{ij}(u)$ for all $i, j \in S$ and $u \in U(i)$ by recording state transitions and actions taken from the past. For example, to model a configuration of a robot we observe its movement sequentially over time. In the system described below we can always observe a state of the real system, and indirectly influence the states the system goes to by taking actions. We call the real system a "Black Box" which consists of the $p_{ij}(u)$. The state $i \in S$ of the system is internal to the "Black Box" but the p-learner can observe it. The p-learner gives the "Black Box" an action $u \in U(i)$ and the "Black Box" produces a new state $j \in S$ according to $p_{ij}(u)$ as illustrated in Figure 2.2 below. Methods for choosing actions are discussed in Section 2.3.2. This system is known as a *passive* system because each state $i \in S$ is observed internally. In Chapter 4 we talk about a system which is *active*, that is when we can intervene by choosing both states and actions.

Since time indices in the p-learner correspond to the system moving forward

in time, it is helpful to call the index of time in the p-learner, time t, even though in principle it may not necessarily denote real-time. An explanation illustrating the relationship between stage k of the computation in the optimiser and time t in the p-learner can be found in Section 2.4.2 below.

The multinomial distribution is a natural model to use when observations have to be classified into a finite number of categories. Its parameters are defined in terms of the probability of an observation falling into any one of these categories. O'Hagan (1994) states that the Dirichlet distribution forms the natural conjugate family to the multinomial likelihood. Let us use the Dirichlet distribution to model the state transitions at each time t. We abuse the notation used by Barto *et al.* (1995) to describe the components involved in estimating the state transition probabilities. For each discrete time t, the current model of the system consists of Dirichlet posterior estimates of the true state transition probabilities for all states $i, j \in S$ and actions $u \in U(i)$, denoted by $\hat{p}_{ij}^t(u)$. Let $x_{ij}^u(t)$ be the observed number of transitions from state i to state j using action u during the time interval [0, t). Let $x_i^u(t) = \sum_{j \in S} x_{ij}^u(t)$ be the number of times the system was in state i and employed action u before step time t. Thus the Dirichlet posterior estimates of the state-transition probabilities at time t are,

$$\hat{p}_{ij}^t(u) = \{x_{ij}^u(t) + 1\} \left\{ \sum_{j \in S} x_{ij}^u(t) + |S| \right\}^{-1}. \tag{2.3.9}$$

The p-learner algorithm is described as follows: initially at $t = 0$, a state $i_0 = i$ is chosen at random from S and all the state transition probability estimates $\hat{p}_{ij}^0(u)$ are set to $|S|^{-1}$. For $t = 0, 1, 2, \ldots$ the p-learner chooses an action $u_t = u$ which is given to the "Black Box". The p-learner then observes the next state of the system $i_{t+1} = j$ as shown in Figure 2.2 below. After the p-learner has gained its information by observing the new state of the process at time t it updates the current state transition probability estimates using Equation (2.3.9) above.

We can think of the Black Box as being a machine, whereby it knows the true state transition probabilities but we do not; and the only way we will learn about

2.3 The P-Learner

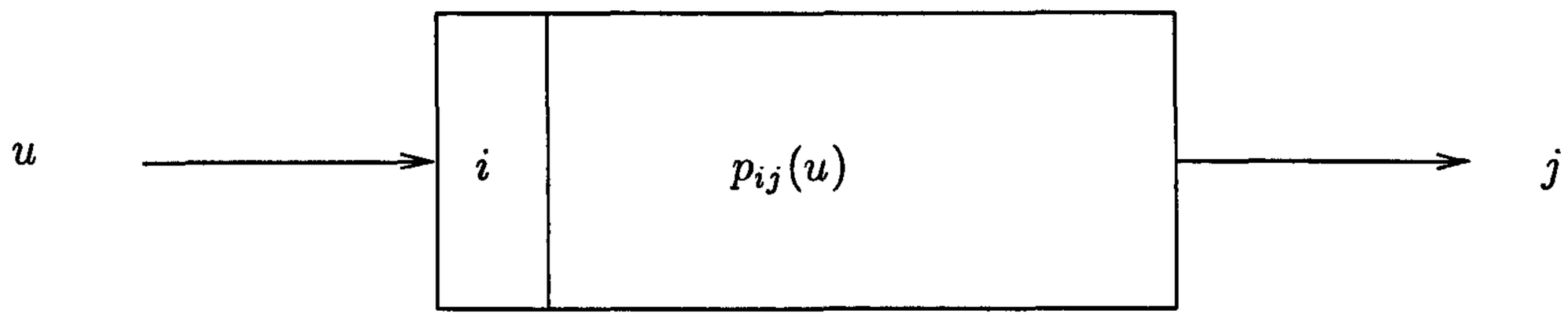


Figure 2.2: The diagram above illustrates how the p-learner simulates the real state transition probabilities by observing the system's current state $i \in S$ and issuing an action $u \in U(i)$ to the "Black Box" to get an update of the process $j \in S$.

the true state transitions is by simulating it. The p-learner must learn about the true $p_{ij}(u)$ otherwise it will not be learning about the right transitions.

It is hoped that the current state transition probability estimates corresponding to each state $i \in S$ and each action $\pi^*(i)$ will eventually converge to their corresponding true state transition probabilities. Given the current action estimates eventually settle down to be the true optimal actions in each state, this would then be sufficient information for the optimiser to be able to find the true optimal value function $v^*(\cdot)$ and the true optimal policy $\pi^*(\cdot)$. The learning functions we present in the next section for choosing actions, which are extensions of previous exploration methods found in John (1995), help us do this. However, time constraints and large state and action spaces may prevent calculations being iterated to convergence due to a lack of exploration.

In Chapters 3 and 4 we look at how closely our current value function estimates and current policy estimates match their corresponding true values. We also use diagnostic tools to look at how close our current model estimates compare to the true model.

In the next section we present learning functions developed for choosing actions used both in the optimiser and the p-learner.

2.3.2 Choosing Actions

This section describes the various learning methods we have developed for choosing actions, in both the optimiser and p-learner, that we will apply in case studies 1 and 2 in Chapters 3 and 4. The learning methods help both the optimiser and the p-learner sample state and action spaces. Other learning methods developed by other authors are discussed in Section 1.7. A variant of Semi-Uniform Distributed Exploration and two variants of Boltzmann Exploration labelled methods 1, 2 and 3 respectively have been developed and are described below. We have indexed the various methods using step time t, but in general if they are to be implemented in the optimiser the index t should be substituted for the index k.

2.3.2.1 Method 1 - A variant of the Semi-Uniform Distributed Exploration Method

This method is an extension of the Semi-Uniform Distributed Exploration Method described in Section 1.7.2.1.

Contrary to the Semi-Uniform Distributed Exploration Method, our method chooses a current action estimate with probability $\rho(t)$ at time t and actions uniformly with probability $1 - \rho(t)$ where $\rho(t)$ is a probability dependent on time t. Thus when in state $i_t = i$ at time t, action $u_t = u$ is chosen using Equation (1.7.28) above except probability p is substituted for probability $\rho(t)$. The function $\rho(t)$ at time t is illustrated in Figure 2.3 below and is defined as,

$$\rho(t) = (1 - \exp(-Wt))M,$$

where W and M are experimental parameter constants. These experimental parameters determine the precise nature of learning. We interpret the parameters as follows: M determines the limiting value of $\rho(t)$ where $0 \leq M \leq 1$, and W governs the function's rate of convergence to M where $6 \times 10^{-5} < W < 0.01$. We

can see from Figure 2.3 that the lower the value of W the slower $\rho(t)$ converges to M. So by varying M and W we can vary the proportion of time the algorithm spends exploring and exploiting. The figure also illustrates that at first $\rho(t)$ is small so that the algorithm can spread its effort evenly over the whole action space, then over time information of the system accumulates and the algorithm focuses in on the most important actions in each state as $\rho(t)$ converges to M.

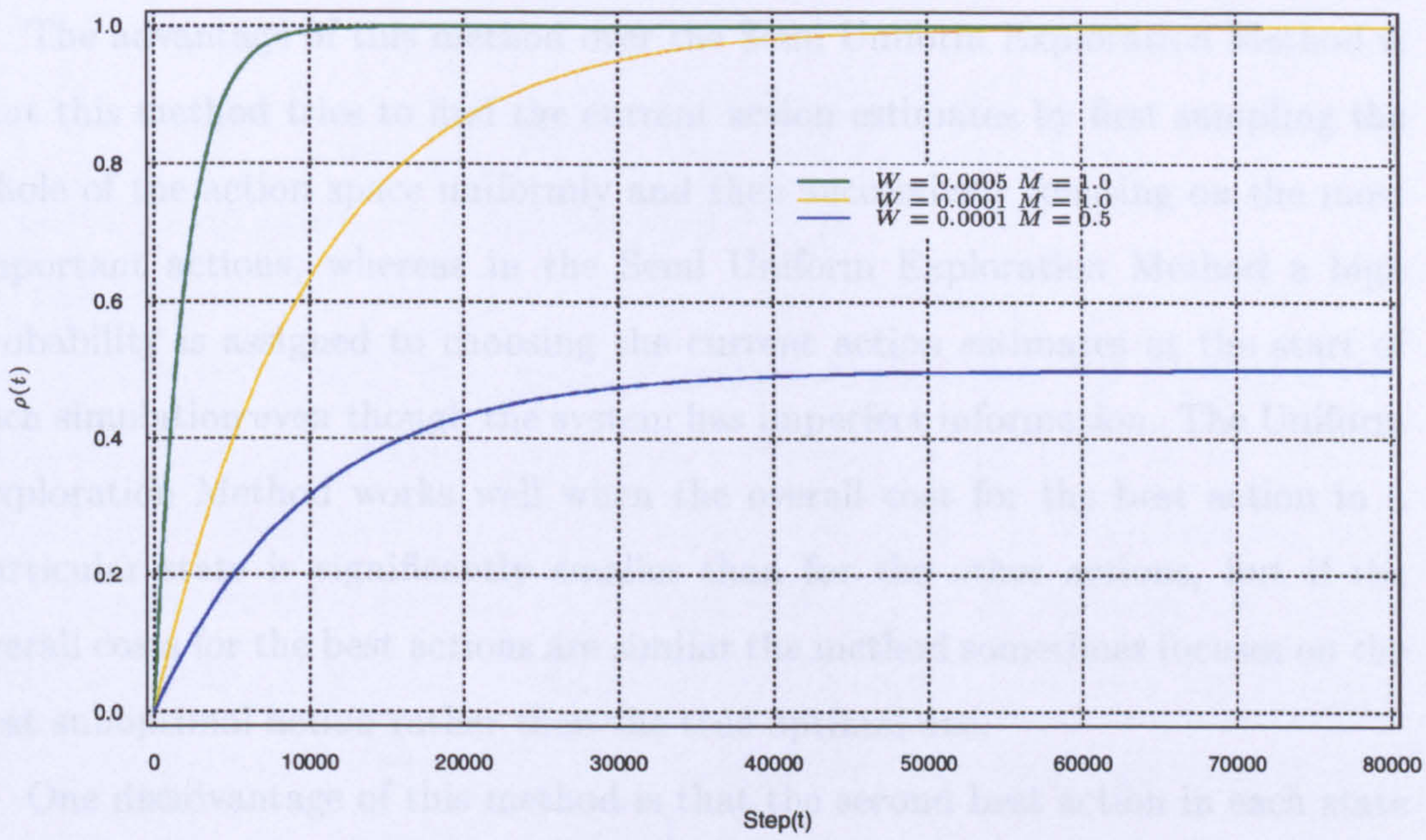


Figure 2.3: The graph above is a plot of the probability function $\rho(t)$ (used to determine which action to take in Method 1) plotted against step time t using learning parameter sets ($W = 0.0005$, $M = 1.0$), ($W = 0.0001$, $M = 1.0$) and ($W = 0.0001$, $M = 0.5$) where $T = 80000$. We experimented with various combinations of values for learning parameters W, M and $T = 80000$, and a good illustration of the effects of different learning parameters can be got by looking at the ones plotted above.

Finding a good learning parameter set (W, M) for a particular problem depends on the size of the state and action spaces given a limited computer budget T. If T is large, having a low bound for M may force the system to spend too much time exploring, while not doing enough exploiting. Thus the algorithm's solution may take an unnecessarily long time to converge to the optimal solution.

A similar argument can be made if, W, the rate of convergence to M, is too slow. Ideally, the bound M should be set to unity so that eventually the important parts of the spaces can be sampled with probability one. However, if T is small the above arguments are reversed, since if the algorithm decides to discriminate between actions too early, the system may settle down to something other than the optimal solution. Therefore as a general rule of thumb: as the computer budget T reduces, a relatively larger fraction of it will be needed to explore.

The advantage of this method over the Semi Uniform Exploration Method is that this method tries to find the current action estimates by first sampling the whole of the action space uniformly and then successively focusing on the most important actions, whereas in the Semi Uniform Exploration Method a high probability is assigned to choosing the current action estimates at the start of each simulation even though the system has imperfect information. The Uniform Exploration Method works well when the overall cost for the best action in a particular state is significantly smaller than for the other actions, but if the overall costs for the best actions are similar the method sometimes focuses on the best suboptimal action rather than the true optimal one.

One disadvantage of this method is that the second best action in each state is eliminated fairly quickly even if the overall cost between the two best actions in a particular state are close. The method also suffers from a trait common to all indirected methods: experimental parameters have to be meticulously tuned for the algorithm's solution to converge to the optimal solution (Kaelbling, Littman, and Moore 1996). A visual example of the disadvantages using this method are illustrated in case study 1 in Section 3.4.

2.3.2.2 Method 2 - A variant of the Boltzmann Exploration Method

This method is an extension of the Boltzmann Exploration Method described in Section 1.7.2.2.

2.3 The P-Learner

In our method we use a similar Boltzmann Exploration Function as in Equation (1.7.29). Let us change notation slightly and denote the current value function estimate when the optimiser is concurrently run with the p-learner to be $\widetilde{v}$ instead of $\widehat{v}$ as in Section 2.2, when the optimiser is run on its own . When in state $i_t = i$ we choose an action $u_t = u$ at time t with probability,

$$\frac{\exp\left\{-\gamma'(t)\omega_t(i,u)\right\}}{\sum_{b\in U(i)}\exp\left\{-\gamma'(t)\omega_t(i,b)\right\}} \quad \text{for all} \quad u \in U(i), \tag{2.3.10}$$

where

$$\omega_t(i,b) = \frac{\widetilde{v}_{t+1}(i,b) - \min_{u\in U(i)}\left\{\widetilde{v}_{t+1}(i,u)\right\}}{\max_{u\in U(i)}\left\{\widetilde{v}_{t+1}(i,u)\right\} - \min_{u\in U(i)}\left\{\widetilde{v}_{t+1}(i,u)\right\}} \tag{2.3.11}$$

at time t for all states $i \in S$ and actions $b \in U(i)$.

The function $\gamma'(t)$ is initially set to zero and it gradually increases with time t to a finite value at T. It is formally defined in Equation (2.3.12) below. On the other hand, the quantity $w_t(i,u)$ is a measure of the belief in the relative advantage of taking action $u \in U(i)$ in state i at time t. It is a weighting function whereby as time goes by we put more weight in to choosing the current action estimate. If action $u = \arg\min_{u\in U(i)}\{\widetilde{v}_t(i,u)\}$ then $w_t(i,u) = 0$ and if action $u = \arg\max_{u\in U(i)}\{\widetilde{v}_t(i,u)\}$ then $w_t(i,u) = 1$. Therefore to begin with when $\gamma'(t)$ is approximately zero, actions $u \in U(i)$ in state i are chosen with equal probability, then as the algorithm gains more information and $\gamma'(t)$ increases the current action estimate in state $i \in S$ is the most desirable action.

We use Equation (2.3.11) because it seems like a sensible initial choice. If $\omega_t(i,b)$ were equal only to the numerator in Equation (2.3.11), the function would depend on the value of the costs functions. By normalising, using the measure of spread in the value function for each state i, takes away the dependence of the cost units, making all the states equal in their performance (dimension free). We cannot exponentiate cost units, only numbers.

The function $\gamma'(t)$ is formally defined as,

$$\gamma'(t) = \gamma(t)\Delta, \tag{2.3.12}$$

where $\Delta < \infty$ is a constant and,

$$\gamma(t) = \frac{\exp\{(t-\mu)/\sigma\}}{1+\exp\{(t-\mu)/\sigma\}}. \tag{2.3.13}$$

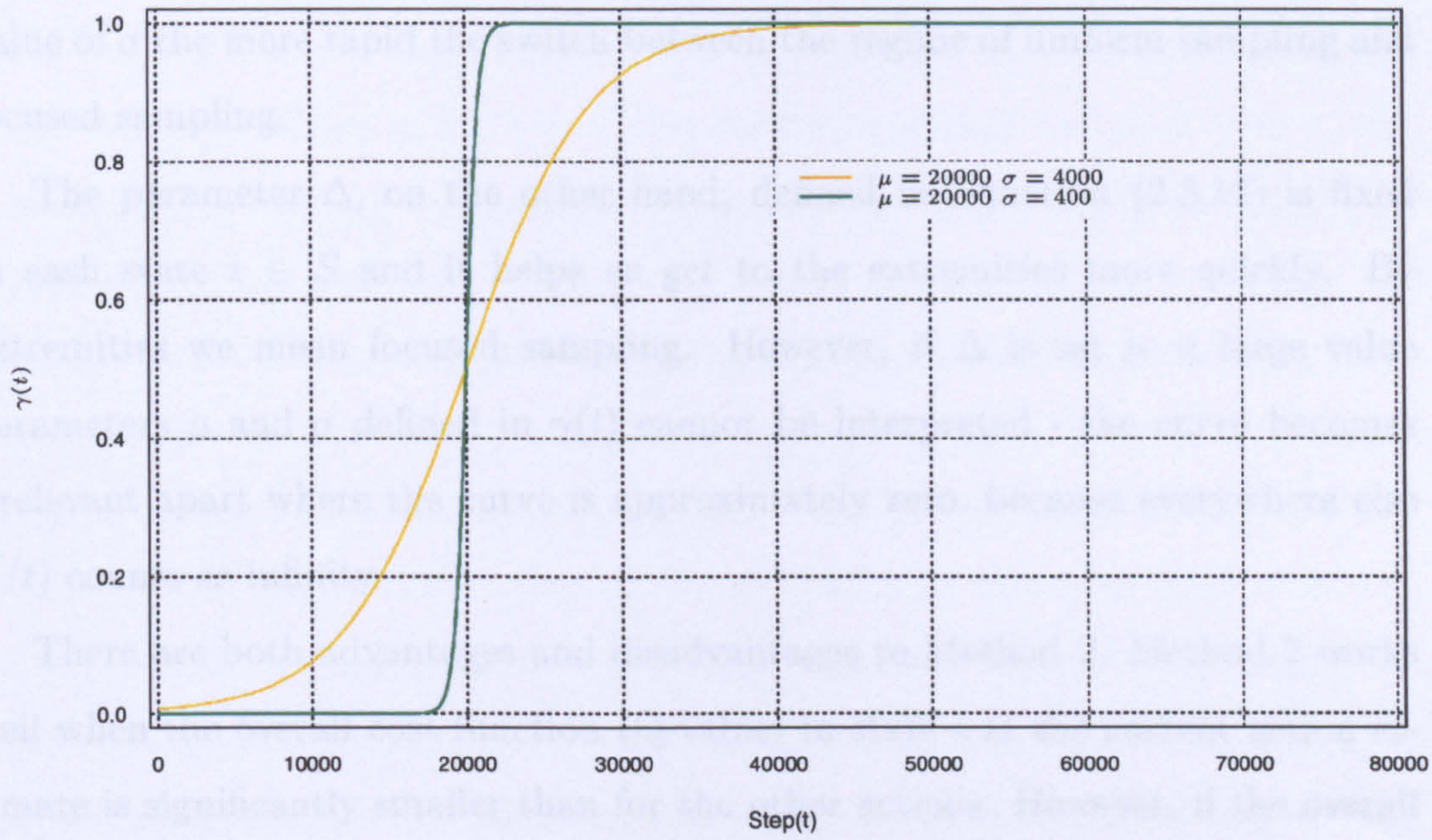


Figure 2.4: The graph above illustrates $\gamma(t)$ being plotted against step time t for learning parameter sets ($\mu = 20000$, $\sigma = 4000$) and ($\mu = 20000$, $\sigma = 400$) where $T = 80000$. We plotted these function for a range of values of these parameters and a typical and a good illustration of how the method works can be got by looking at this particular set of parameters.

The curve $\gamma(t)$ is illustrated in Figure 2.4 above for different learning parameter sets. The curve depends on two parameters, μ and σ. The parameter μ is defined as the amount of learning we are willing to do. It is the point at which we switch from the mode of sampling over all actions in each state (exploring) to focused sampling (exploiting). The parameter σ, on the other hand, defines the steepness of the slope which determines how rapidly we switch from exploring to exploiting. Both of these parameters can be altered to gain sensitive control over the system.

The parameter μ is a location in T at which the function $\gamma(t)$ takes the value 0.5. It is the location of the steepest part of the slope. If we increase μ the

steepest part of the curve moves of the right and if we decrease μ the steepest part of the curve moves to the left. Thus the bigger the value of μ the longer we sample over all the actions. Likewise, if we increase σ the curve gets flatter and if we decrease σ the curve gets steeper (see Figure 2.4). Thus, the lower the value of σ the more rapid the switch between the regime of uniform sampling and focused sampling.

The parameter Δ, on the other hand, defined in Equation (2.3.12) is fixed in each state $i \in S$ and it helps us get to the extremities more quickly. By extremities we mean focused sampling. However, if Δ is set to a large value parameters μ and σ defined in $\gamma(t)$ cannot be interpreted - the curve becomes irrelevant apart where the curve is approximately zero, because everywhere else $\gamma'(t)$ counts as infinity.

There are both advantages and disadvantages to Method 2. Method 2 works well when the overall cost function (Q-value) in state i at the current action estimate is significantly smaller than for the other actions. However, if the overall costs for the best actions in state i are close the method will on occasions experience difficulties when trying to focus on the true optimal action in state i, and instead it will opt for the best suboptimal action. This is true for all Boltzmann Exploration type methods (Kaelbling, Littman, and Moore 1996). Also, if a bad combination of values for parameters μ, σ, Δ and T are chosen our solutions will converge very slowly to the true solution of the problem or may not even settle down to it at all. A similar argument can be made for the Boltzmann Exploration Method if the temperature schedule Z_t is not tuned in properly. In addition, we found that when $\Delta = 1$, we saw it was not possible to ensure that the probability (defined in Equation (2.3.10) above) of choosing a current action estimate in each state converged to unity as $t \to \infty$, if $\gamma'(t)$ was bounded. An example of this is shown in case study 1 in Section 3.4. We then looked at the reason for using Δ in Method 2 and realised we were mistaken in using a function $\gamma'(t)$ with an upper threshold. This is because all we were doing by changing Δ was changing

the asymptote of $\gamma'(t)$. This has helped us to develop Method 3.

2.3.2.3 Method 3 - A another variant of the Boltzmann Exploration Method

As in Method 2 above this method is also an extension of the Boltzmann Exploration Method. When in state $i_t = i$ we choose an action $u_t = u$ at time t with a probability according to Equation (2.3.10) above. However, the function $\gamma'(t)$ is defined slightly differently than in Method 2. Unlike Method 2, $\gamma'(t)$ is not bounded but is still defined in terms of the same parameters μ and σ. The function itself is a smooth curve to help us control the system better. The function starts off at zero and gradually increases until time μ, allowing us the opportunity to learn about the overall system. After time μ, $\gamma'(t)$ increases linearly in t to infinity, so that we can focus in on the most important states and actions. Note there is no special reason for $\gamma'(t)$ to increase linearly after time μ. The use of an exponential function was considered but as an exponent increases very rapidly with time this may cause us to discriminate between actions too early and focus on suboptimal actions in certain states.

The function $\gamma'(t)$ at time t is defined as,

$$\gamma'(t) = \Delta\sigma\gamma(t), \tag{2.3.14}$$

where,

$$\gamma(t) = \max\left\{0, \frac{t-\mu}{\sigma}\right\} + \exp\left\{-\left|\frac{t-\mu}{2\sigma}\right|\right\}, \tag{2.3.15}$$

The curve $\gamma(t)$ has the property that as $t \to -\infty$, $\gamma(t)$ asymptotes to 0, and as $t \to \infty$, $\gamma(t)$ asymptotes to $\frac{t}{\sigma}$. Therefore if we had set $\gamma'(t) = \gamma(t)$, σ would have had two qualitative effects; it would have described the asymptotic slope as $t \to \infty$ and the sharpness of the curve near time μ, both of which are clearly different. However, multiplying $\gamma(t)$ by σ and Δ lets σ refer to the sharpness of the curve between the regime of uniform sampling over all the actions and

focused sampling, and Δ refer to the asymptotic slope after time μ. An example of the effects of μ, σ and Δ is illustrated in Figure 2.5 below.

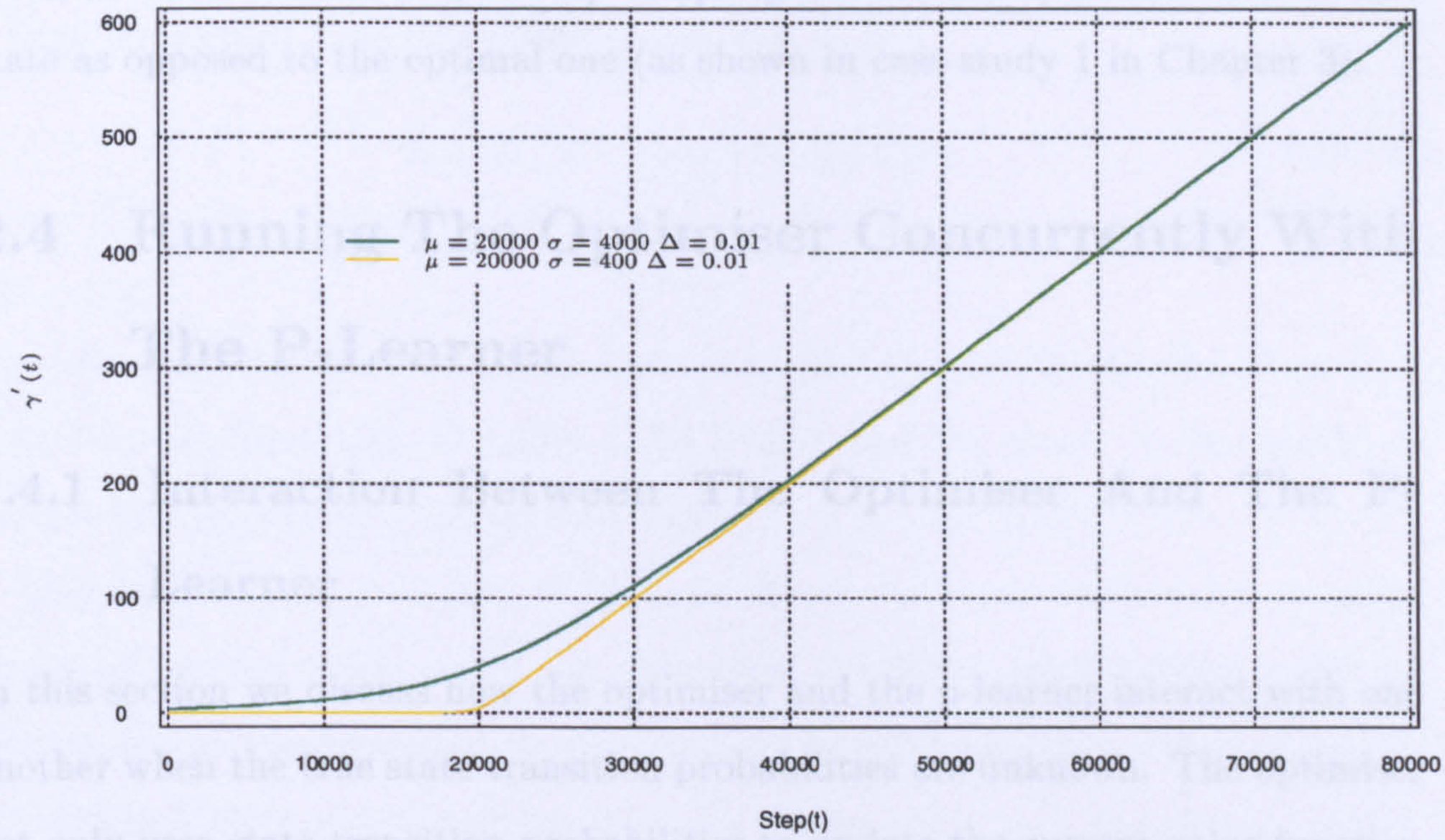


Figure 2.5: The above diagram illustrates the qualitative importance of the learning parameters for $\mu = 20000$, $\sigma = 400$ or 4000 and $\Delta = 0.01$. The figure shows the bigger the value of σ the smoother the curve thus uniform sampling is not achieved for very long but the transition between uniform sampling and focused sampling is somewhat delayed.

If σ and Δ are set to very small values in Equation (2.3.14) the curve $\gamma'(t)$ will be very close to the asymptote as illustrated in Figure 2.5 and uniform sampling will be achieved up to point near time μ, while swiftly moving to focused sampling. On the other hand, if σ is large and Δ is small the curve $\gamma'(t)$ will rise immediately above zero so that uniform sampling only lasts for a short time, but the transition between uniform and focused sampling takes a little longer.

The advantage of Method 3 over Method 2 is that this method ensures we get to the extremities, that is, the probability of sampling the current action estimate as time goes on in a particular state tends to 1 and the others tend to 0. This

happens even if the overall costs for two actions in the same state are fairly close. However, one disadvantage, as with all indirected methods is that if we use a bad choice of learning parameters and the overall costs between actions are fairly close, then the method may end up sampling the best suboptimal action in that state as opposed to the optimal one (as shown in case study 1 in Chapter 3).

2.4 Running The Optimiser Concurrently With The P-Learner

2.4.1 Interaction Between The Optimiser And The P-Learner

In this section we discuss how the optimiser and the p-learner interact with one another when the true state transition probabilities are unknown. The optimiser not only uses state transition probabilities to update the current value function estimates and the current policy estimates, but it also uses them to generate states. When the true state transition probabilities are unknown the p-learner is used to estimate them, but we must ask how the optimiser uses the information given to it by the p-learner? We first describe how the optimiser uses these estimates to update the current value function estimates and the current policy estimates and then how it generates states.

DP requires an accurate model of the system. Immediate costs $c_i(u)$ are available in a look up table and the state transitions are estimated via the p-learner. Thus when the optimiser is run concurrently with the p-learner we set up a surrogate model replacing the true state transition probabilities with our estimates in Equation (2.2.4). Assuming an initial estimate $\widetilde{v}_0(\cdot)$, a sequence of value functions estimates $\widetilde{v}_{k+1}(\cdot)$ and policy estimates $\widetilde{\pi}_{k+1}(\cdot)$, are generated at

each stage $k \geq 0$ using intermediate functions,

$$\widetilde{v}_{k+1}(i,u) = \begin{cases} c_i(u) + \alpha \sum_{j \in S} \hat{p}^t_{ij}(u) \widetilde{v}_k(j) & \text{if } i \in S_k, \\ \widetilde{v}_k(i,u) & \text{if } i \notin S_k, \end{cases} \tag{2.4.16}$$

where, unchanged from Equations (2.2.2) and (2.2.3):

$$\widetilde{v}_{k+1}(i) = \min_{u \in U(i)} \{\widetilde{v}_{k+1}(i,u)\}, \tag{2.4.17}$$

$$\widetilde{\pi}_{k+1}(i) \in \arg\min_{u \in U(i)} \{\widetilde{v}_{k+1}(i,u)\}. \tag{2.4.18}$$

Note the only difference between Equations (2.2.4), (2.2.5) and (2.2.6) in Section 2.2 and Equations (2.4.16), (2.4.17) and (2.4.18) above is that $p_{ij}(u)$ is replaced by $\hat{p}^t_{ij}(u)$ and $\widehat{v}$ is replaced by $\widetilde{v}$.

Similarly in the optimiser, if we were to use the same method for choosing states as Section 2.2.2, again the true state transition probabilities would be substituted for the estimated state transition probabilities. Thus Equation (2.2.7) in Section 2.2.2 would change to,

$$q_{ij}(u) = \hat{p}^{1/T_k}_{ij}(u) \left\{ \sum_{l \in S} \hat{p}^{1/T_k}_{il}(u) \right\}^{-1}, \tag{2.4.19}$$

for all states $i, j, l \in S$ and actions $u \in U(i)$, where i is the current state, j is the proposed state, l is the possible successor state and T_k is the temperature schedule at stage k.

In the following sections we show how the two algorithms for the optimiser and the p-learner are implemented.

2.4.2 Running The Optimiser And The P-Learner In Parallel

The remaining sections in this chapter discuss reasons behind why and how the optimiser and p-learner are run in parallel with each other. This section discusses why they are run in parallel with each other. We have already established that

the optimiser and the p-learner collectively solve the problem of optimising the cost of a system given imperfect information, but why are they run in parallel? The way we sample transitions when we know the true values of the system parameters is not necessarily applicable when we do not know these values. This is because there are distinct problems in deciding how to a) sample calculations in the optimiser and b) sample transitions in the p-learner; and the two problems are independent. Thus the optimiser and p-learner can be run in parallel with each other.

One of the advantages of running the optimiser and p-learner in parallel is that we are not restricted to running both at every iteration. For instance, we may want to leave the optimiser alone for a while to learn about the system parameters before we do any more optimising, then when we decide to do a bit more optimising, the optimiser is still in the same state and stage we had previously left it in, or vice versa. In many problems it may be the case that the system needs several updates of one between updates of the other. Another advantage of keeping both algorithms apart is that it gives us more freedom to choose our own strategies and does not restrict the learning process to a particular path, thus enabling us to explore a different range of algorithms and giving us a good mixture of information about the states and transitions. In addition, the learning that we do is not biased.

Please note that in this thesis the two algorithms are run simultaneously at each iteration. Therefore for the purpose of representing results in Chapters 3 and 4 we refer to both sets of indices as time t (even though in our system the indexing of the optimiser could be done in a slightly different way using stages k).

2.4.3 Access To Information

The following section describes the optimiser's facility to access information. Throughout this section we refer to the architecture of the algorithm illustrated in Figure 2.6 below. In Figure 2.6 an arrow pointing towards the optimiser or the p-learner denotes permission to read data whereas an arrow pointing away from either of the two denotes permission to write to existing data (update). If the optimiser is run on its own (system parameters known) we only consider the optimising part of Figure 2.6 and if the optimiser and the p-learner are run concurrently (system parameters unknown) we consider both the optimising and learning parts of Figure 2.6.

At all times the optimiser has read access to the immediate cost matrix C which consists of elements $c_i(u)$ and the matrix P, be it the true state transition probabilities $p_{ij}(u)$ (KNOWN P) or the current estimates $\hat{p}^t_{ij}(u)$ (UNKNOWN P) at time t for all $i \in S$ and $u \in U(i)$ (note the arrow points in only the one direction in Figure 2.6, towards the optimiser). When the optimiser is run on its own the known matrix P is used and when run concurrently with the p-learner the unknown matrix P is used. The optimiser has both read and write access to the vector of the current value function estimate (denoted by V) but only write access to the vector of the current policy estimate (denoted by Π). The optimiser uses C, P, and V to continually update V and Π using Equations (2.2.4), (2.2.5) and (2.2.6) if the optimiser is run on its own or Equations (2.4.16), (2.4.17) and (2.4.18) if it is run concurrently with the p-learner.

The p-learner on the other hand has limited (readable) access to the true state transition probabilities denoted by *Real* P which it uses to simulate transitions by observing states and feeding actions into the Black Box to obtain an update of the process. The p-learner sometimes uses the vector V in its exploration functions for choosing actions (Methods 2 and 3) and therefore has read access only to vector V (see Equations (2.3.10) and (2.3.11)). The p-learner records the

transitions made and updates the current state transition estimates (UNKNOWN P) which in turn is given to the optimiser to update V and Π.

The listing of the main algorithm is listed in the next section below.

2.4.4 The Computation

This section illustrates how the optimiser and the p-learner are run in parallel by means of listing the main algorithm below. The main algorithm consists of four lines and it runs from time $t = 0$ to time T. We present the algorithm in the form of *C code*:

```
line (1)    for(t=0; t < T; t++)                    }
line (2)            optimiser(C, P, V, Π, t)
line (3)            p-learner(RealP, P, V, t)
line (4)    }
```

Note if lines 2 and 3 are included in the main algorithm, we can solve the stochastic cost problem by estimating the state transition probabilities in order to do the optimisation. If line 3 is deleted we are back to the problem of optimising given the true values of the system parameters. An illustration of the design of the algorithm can be found in Figure 2.6 below.

2.5 Applications

There are countless applications that can be formulated as stochastic optimal control problems, where system parameters are not necessarily known. In Chapters 3 and 4 for comparison purposes a large computer budget was introduced, but in real life applications such large computer budgets may not be applicable. Two possible realistic examples are illustrated below. The first is an example of condition monitoring and the second is an example of robotic control.

Aircraft engine components have to be monitored at certain fixed time periods. Components deteriorate over time and a decision has to be made whether to

replace them with new ones. Neglecting to monitor the condition of components may cause the aircraft to break down. The life expectancy of a component on an aircraft can be modelled using a Weibull distribution. Life components can be put into two main categories: major components; such as engine rings and main bearings, and minor components; such as a gearboxes and hydraulic pumps. If major components malfunction the engine is taken to the overhaul shop for lengthy repairs. On the other hand if minor components fail, they can be replaced with the engine still on the wing. Keeping the aeroplane on the ground costs money. The aim is to minimise the length of time spent on the ground between scheduled maintenance checks.

In this problem the state space is the condition of the engine. The costs incurred are the time lost while the aeroplane is on the ground, maintenance costs, the running costs of the engine (dependent on its condition) and replacement costs. The controller has a choice of three actions; to "leave", "repair" or "replace" the engine. A simple example of this problem is illustrated in Chapter 4.

Robots are used in hostile environments where direct or remote control by humans is not practical, for example when surveying the Antarctic, deep sea, volcanos or the surfaces of other planets. In these cases the robot moves around near a central base to which it must return periodically in order to recharge its batteries and transfer data. The large, static base will transmit the data back to the scientists and may contain solar panels or have large batteries. The robot will move around at random (or on a course determined by a random starting direction) until it returns to base again.

In this application the state space comprises the Euclidean distance of the robot from the base and the charge left on the batteries. The robot has two actions; "return to base" or "continue collecting data". The costs incurred are the time lost owing to return and recharging, and the cost of unit failure due to insufficient power. Other robot and control applications are discussed in Kaelbling *et al.* (1996).

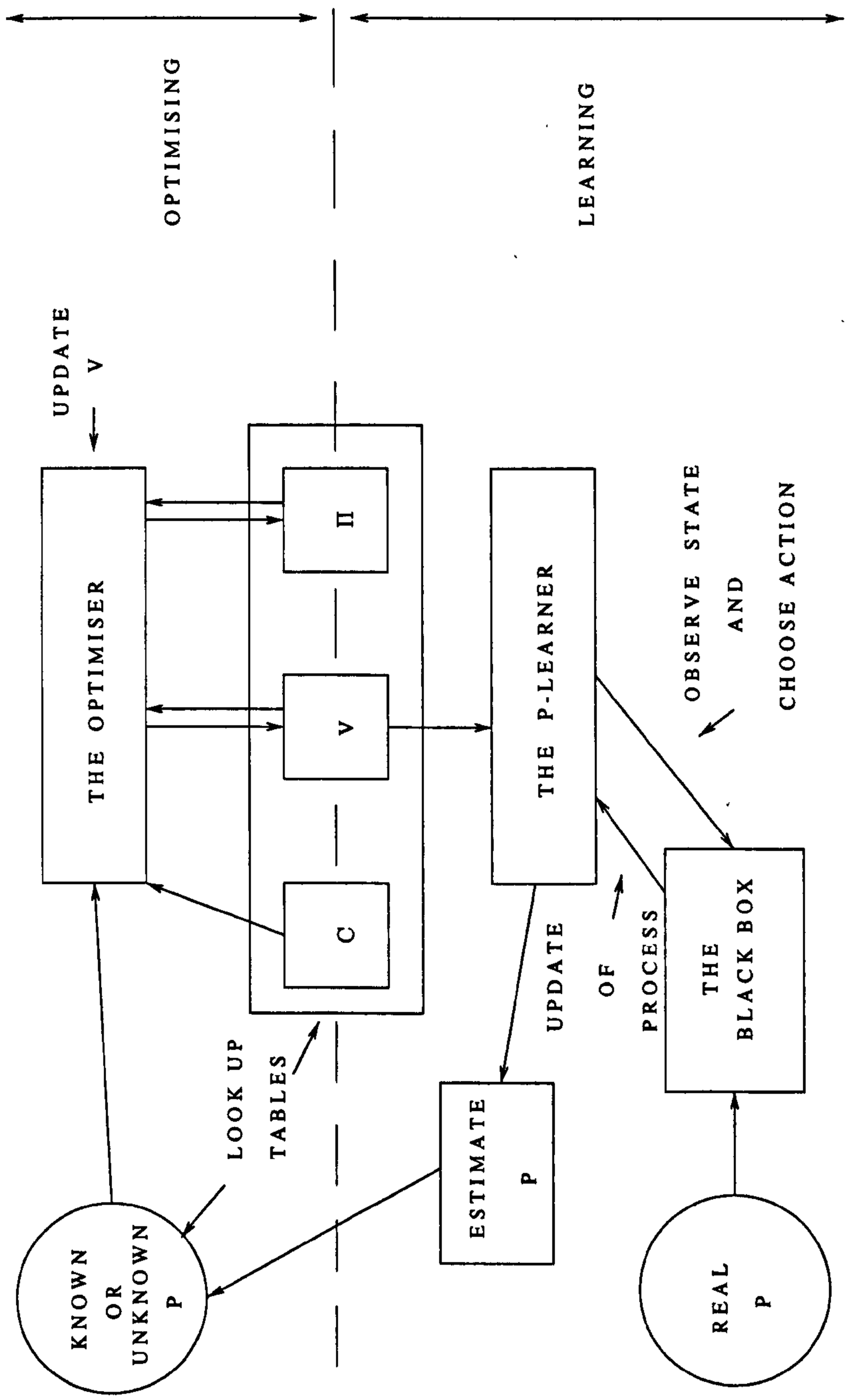


Figure 2.6: The above diagram is the architecture of the two processes: the optimiser run on its own (the known P case) where we only consider the optimising part of the diagram; and the optimiser concurrently run with the p-learner (the unknown P case) where we consider both the optimising and learning parts of the diagram simultaneously.

Chapter 3

CASE STUDY 1 : A Simple Fully Connected Problem

3.1 Introduction

This chapter considers a simple discounted cost minimisation problem described in Section 3.2 below. We use it to illustrate the performance of the optimiser when run both on its own and concurrently with the p-learner, in each case comparing the performance with that of the Pre-Jacobi method. The criteria we will use relate to their ability to find a true optimal policy and the true optimal value function in each state $i \in S$ using a sequence of simulations of fixed size T.

After describing the illustrative problem, we review the methodology and we discuss reasons why we focused on Method 3 in Section 2.3.2 instead of Methods 1 and 2. We then describe the choice of learning parameters μ, σ and Δ used in our study and in detail the procedure at each iteration both in the optimiser and the p-learner, so as to precisely define the various measures of success we wish to discuss in the rest of the chapter. We introduce and outline the various criteria we will use to evaluate performance across different combinations of learning parameters, based on how closely our current value function estimates,

our current policy estimates and our current model estimates compare to their corresponding true values. We first analyse the results obtained by running the optimiser is run on its own. These results are then used as a reference point for when the optimiser is run concurrently with the p-learner.

3.2 Problem Description

A simple infinite horizon cost minimisation problem with a discount factor α was set up to compare the performance of the two methods we developed with the more conventional Pre-Jacobi method. The problem consists of ten states labelled from 0 to 9 and allows three actions in each state, labelled from 0 to 2. The immediate cost matrix defined for each state and action is shown in Table 3.1 below and the set of state transition probabilities defined for each state and action can be found in Appendix A.

Actions	States i									
$u \in U(i)$	0	1	2	3	4	5	6	7	8	9
0	350	300	200	400	800	900	300	600	100	500
1	300	245	170	150	150	150	200	500	90	400
2	200	190	100	100	120	100	100	200	80	100

Table 3.1: The cost matrix for each state 0 to 9 and actions within states 0 to 2.

Our algorithms to some extent are based on value iteration. For known system parameters value iteration methods take longer to converge to the exact solution when α is close to 1. A discount factor of $\alpha = 0.9$ was therefore used because it might be expected to provide a more testing situation to evaluate our algorithms.

The system parameters such as the immediate costs and the state transition probabilities were also carefully chosen. We chose all the state transition probabilities to be non-zero because they are representative of the problem studied by White (1963). He calls this type of generic problem the "completely ergodic"

problem since it is possible to move, in a single time step, from a state $i \in S$ to any state $j \in S$ given any action $u \in U(i)$. This means whatever policy we choose to use in our algorithms, all the states are fully connected and as a result rapid mixing between states will be present. The immediate costs $c_i(u)$ on the other hand, for each $i \in S$ and $u \in U(i)$, were chosen in conjunction with the positive state transition probabilities to give us a broad range of behaviour in the system across different states. This can be seen by looking at the optimal "Q-values" in Table 3.2 below.

In Section 1.2.5 we defined the optimal Q-value function $Q^*(i,u)$ to be the optimal value function v^* evaluated at a state i and action $u \in U(i)$. Thus,

$$Q^*(i,u) = c_i(u) + \alpha \sum_{j \in S} p_{ij}(u) v^*(j) \quad \text{for all } i \in S \text{ and } u \in U(i),$$

$$v^*(i) = \min_{u \in U(i)} \{Q^*(i,u)\} \quad \text{for all } i \in S,$$

$$\pi^*(i) \in \arg\min_{u \in U(i)} \{Q^*(i,u)\} \quad \text{for all } i \in S.$$

Table 3.2 below is a table of the optimal Q-values $Q^*(i,u)$ and the optimal actions $\pi^*(i)$ for this particular problem, defined for each state $i \in S$ and action $u \in U(i)$. If we looked at a one step problem and took the current action estimate from then on, in the hope of identifying optimal actions in different states, from Table 3.2 we can see that for some states it is clear which action is best but for others the best action is not so obvious. This has motivated us to colloquially refer to states $\{0,1,2,3,6,7,9\}$ as the "easy states", states $\{4,5\}$ as the "hard states" and state $\{8\}$ as the "very hard state". When we apply our algorithms to this problem we expect to see difficulties in states $4,5$ and 8 especially when the true state transition probabilities are unknown.

State $i \in S$	$Q^*(i,u)$ $u=0$	1	2	$\pi^*(i)$
0	1498.929	1421.407	**1341.166**	2
1	1426.104	1396.954	**1318.535**	2
2	1338.921	1313.615	**1229.388**	2
3	1521.048	1283.250	**1230.372**	2
4	1948.298	1263.140	**1254.341**	2
5	2031.011	1275.058	**1242.126**	2
6	1422.257	1338.430	**1212.976**	2
7	1733.260	1627.114	**1342.630**	2
8	1240.331	**1225.870**	1228.356	1
9	1626.414	1528.621	**1213.414**	2

Table 3.2: The table above presents the optimal Q-values, $Q^*(i,u)$ for this particular problem defined for each state $i \in S$ and action $u \in U(i)$, and the corresponding optimal actions $\pi^*(i)$ defined for each state $i \in S$. These values were obtained using value iteration. Note, all of the bold figures in the above table correspond to the true optimal value function $v^*(i)$ defined for each state $i \in S$.

3.3 The Methodology Used In This Chapter

3.3.1 Review

In this section we briefly review the methodology used in the optimiser and the p-learner, previously described in Chapter 2.

Essentially, the optimiser is used to approximate the optimal value function $v^*(\cdot)$ and an optimal policy $\pi^*(\cdot)$, and the p-learner is used to estimate the unknown system parameters, namely the state transition probabilities. The other system parameters such as the immediate costs are assumed known. If the state transition probabilities are known (the known P case) the optimiser can be run on its own; when they are unknown (the unknown P case) the optimiser is run concurrently with the p-learner. In the known P case the optimiser uses the true state transition probabilities to estimate $v^*(\cdot)$ and $\pi^*(\cdot)$; in the unknown P case the optimiser uses the estimates of the true transition probabilities obtained from

the p-learner in place of the true values to approximate $v^*(\cdot)$ and $\pi^*(\cdot)$.

In Chapter 2 we indexed time steps in the optimiser as stages k and the p-learner as time steps t. However, as explained in Section 2.4.2, because in the unknown P case we run both the optimiser and the p-learner once at each iteration, it is easier to denote the index of time in the two methods as time steps t.

In this chapter we consider the performance of our algorithms over a fixed horizon T. In the next chapter we will introduce and evaluate a dynamic stopping procedure under which the optimiser and p-learner stop once the current value function estimate is close enough to $v^*(\cdot)$.

We first describe what happens at each iteration when the optimiser is run on its own and then when the optimiser is concurrently run with the p-learner.

3.3.2 Running the optimiser on its own

When the optimiser is run on its own the immediate costs $c_i(u)$ and the true state transition probabilities $p_{ij}(u)$ are assumed known for each state $i, j \in S$ and action $u \in U(i)$.

The choice of the initial values is somewhat arbitrary and, as we would expect, does not seem to affect the operation of the algorithms. In practice, at time $t = 0$, we set the initial value function estimates $\widehat{v}_0(l) = 0$ for each state $l \in S$; we set the action estimates $\widehat{\pi}_0(l) = 2$ for each state $l \in S$ and at each run we choose an initial state i_0 at random from S.

At each time t we assume we have information about the current state $i_t = i$, the immediate costs $c_i(u)$ and the set of true state transition probabilities $p_{ij}(u)$ for all states $i, j \in S$ and actions $u \in U(i)$, the current value function estimate $\widehat{v}_t(l, u)$ evaluated at each state $l \in S$ and each action $u \in U(l)$, and the current value function estimate $\widehat{v}_t(\cdot)$.

At each iteration t, the optimiser updates $\widehat{v}_{t+1}(l,u)$ at all states $l \in S$ and actions $u \in U(l)$ using the Asynchronous DP algorithm defined in Equation (2.2.4). The optimiser calculates $\widehat{v}_{t+1}(i,u)$ using the true state transition probabilities $p_{ij}(u)$, the immediate costs $c_i(u)$, and the current value function estimates $\widehat{v}_t(\cdot)$, while the other value function estimates are updated by setting $\widehat{v}_{t+1}(i',u) = \widehat{v}_t(i',u)$ at states $i' \neq i$. Having calculated $\widehat{v}_{t+1}(l,u)$ for all states $l \in S$ the optimiser simultaneously updates the current value function estimate $\widehat{v}_{t+1}(\cdot)$ and the current policy estimate $\widehat{\pi}_{t+1}(\cdot)$ by setting $\widehat{v}_{t+1}(l) = \min_{u \in U(l)} \{\widehat{v}_{t+1}(l,u)\}$ and $\widehat{\pi}_{t+1}(l) = \arg\min_{u \in U(l)} \{\widehat{v}_{t+1}(l,u)\}$ for each state $l \in S$, as defined in Equations (2.2.5) and (2.2.6) respectively. It then chooses the current action $u_t = u$ using the exploration function defined in Section 3.3.4 below.

Having identified the current state-action pair (i,u), the optimiser uses Equation (2.2.7) to choose the next state $i_{t+1} = j$ at which to update the value function estimate. Equation (2.2.7) powers up and re-normalises the row in the true state transition matrix corresponding to state i and action u using the temperature schedule $1/T_t$, where the temperature T_t is defined in Equation (2.2.8). At first the temperature is very high to ensure that the optimiser samples each state uniformly, then as time goes on the temperature decreases allowing the optimiser to focus in on the most important states; the states where the optimising really matters. The optimiser can do this since the process is being controlled to be approximately optimal using the exploration functions defined in Section 3.3.4 below. Then having visited each state often enough it is hoped that $\widehat{v}_{t+1}(i)$ and $\widehat{\pi}_{t+1}(i)$ will eventually converge to $v^*(i)$ and $\pi^*(i)$ respectively for all states $i \in S$ as t tends T.

3.3.3 Running the optimiser concurrently with the p-learner

When the optimiser is run concurrently with the p-learner the immediate costs $c_i(u)$ are assumed known but the true state transition probabilities $p_{ij}(u)$ are

assumed unknown for each state $i, j \in S$ and action $u \in U(i)$. Please note that when the optimiser and p-learner are run concurrently, the sequence of states and actions generated by the p-learner may be different to the sequence of states and actions generated by the optimiser.

In practice, at time $t = 0$ in the optimiser, we set the initial value function estimates $\widetilde{v}_0(l) = 0$ for each state $l \in S$; we set the action estimates $\widetilde{\pi}_0(l) = 2$ for each state $l \in S$; we set the state transition probability estimates $\hat{p}^0_{lm}(u) = |S|^{-1}$ for each state $l, m \in S$ and action $u \in U(l)$, and at each run we choose an initial state i_0 at random from S.

In the optimiser at each time t we assume we have information about the current state $i_t = i$, the immediate costs $c_i(u)$ and the set of current state transition probability estimates $\hat{p}^t_{ij}(u)$ for all states $i, j \in S$ and actions $u \in U(i)$, the current value function estimate $\widetilde{v}_t(l, u)$ evaluated at each state $l \in S$ and each action $u \in U(l)$, and the current value function estimate $\widetilde{v}_t(\cdot)$.

At each iteration t, the optimiser updates $\widetilde{v}_t(l, u)$ at all states $l \in S$ and actions $u \in U(l)$ using the Asynchronous DP algorithm defined in Equation (2.4.16). Equation (2.4.16) calculates $\widetilde{v}_{t+1}(i, u)$ using the current state transition probability estimates $\hat{p}^t_{ij}(u)$, the immediate costs $c_i(u)$, and the current value function estimates $\widetilde{v}_t(\cdot)$, while the other value function estimates are updated by setting $\widetilde{v}_{t+1}(i', u) = \widetilde{v}_t(i', u)$ at states $i' \neq i$. Subsequently, the optimiser simultaneously updates the current value function estimate $\widetilde{v}_{t+1}(\cdot)$ and the current policy estimate $\widetilde{\pi}_{t+1}(\cdot)$ by setting $\widetilde{v}_{t+1}(l) = \min_{u \in U(l)} \{\widetilde{v}_{t+1}(l, u)\}$ and $\widetilde{\pi}_{t+1}(l) = \arg\min_{u \in U(l)} \{\widetilde{v}_{t+1}(l, u)\}$ for each state $l \in S$, as defined in Equations (2.4.17) and (2.4.18) respectively. As with the known P case above, the optimiser then chooses the next action $u_t = u$ according to the exploration function defined in Section 3.3.4. Once the current state-action pair (i, u) has been identified the next state i_{t+1} is chosen using Equation (2.4.19). This state is then used to update the value

function estimate at the next iteration. Equation (2.4.19) powers up and renormalises the estimated transition matrix corresponding to state i and action u using a temperature schedule $1/T_t$ - the temperature schedule is employed for the same reason as in the known P case above.

Having updated the current value function estimate and the current policy estimate in the optimiser we then proceed to update the current state transition probabilities in the p-learner.

In the p-learner at each time t, we assume we have information about the current state $i_t^{'} = i$, the set of current state transition probability estimates $\hat{p}_{lj}^t(u)$ for all states $l, j \in S$ and actions $u \in U(l)$ and the current value function estimate $\widetilde{v}_t(l, u)$ evaluated at each state $l \in S$ and each action $u \in U(l)$.

In practice at time $t = 0$ we set the components involved in estimating the state transition probabilities $x_{lm}^u(t) = 0$, for each state $l, m \in S$ and actions $u \in U(i)$, and we choose an initial state $i_0^{'}$ at random from S. The parameter $x_{lm}^u(t)$ is defined as the number of transitions the system has made from state l to state m given it has used action $u \in U(l)$ before time t.

At each time t the p-learner assumes information about the current state $i_t^{'} = i$, the current value function estimate $\widetilde{v}_{t+1}(l, u)$ evaluated at each state $l \in S$ and each action $u \in U(l)$, and the components, $x_{lm}^u(t)$, defined for each state $l, m \in S$ and actions $u \in U(i)$.

Having observed a state $i_t^{'} = i$ of the system at time t the p-learner chooses an action $u_t^{'} = u$ according to the same exploration function as the optimiser, defined in Section 3.3.4 below. The p-learner subsequently gives the action u to the "Black Box" and the "Black Box" draws a new state $i_{t+1}^{'} = j$ with probability $p_{ij}(u)$ (see Figure 2.2). The transition from state i to state j using action $u \in U(i)$ is recorded and the true state transition probability estimates are then updated via the Dirichlet posterior estimates defined in Equation (2.3.9) above. These estimates are immediately reported to the optimiser in time for the next iteration.

3.3.4 Choosing Actions

In this section we briefly re-cap the method used to choose actions in our simulations. In principal, the actions u_t and u_t' chosen by the optimiser and the p-learner at time t respectively could be chosen according to any of the methods described in Section 2.3.2. However, we will choose actions according to the method labelled 3 described in Section 2.3.2.

The optimiser and p-learner learn about the system by sampling and observing states respectively and by taking action. The algorithms cannot continually sample all of the state-action pairs because we only have a fixed simulation horizon T. Method 3 was therefore designed for the algorithms to spread their effort evenly over the whole state and action spaces in order to identify the optimal policy, then having identified the optimal policy they use it to identify the correct value function. The p-learner tries to repeatedly sample each element in the state transition probability matrix corresponding to each state $i \in S$ and each action $\pi^*(i)$, in order to help the optimiser gain good estimates of the true optimal value function. The optimiser on the other hand tries to concentrate its computational effort on states resulting from an optimal policy because the other states are highly unlikely to ever be visited, especially in large state spaces.

Below we recall the relevant equations from Method 3, writing them in terms of the p-learner. The value function estimate at time t evaluated at each state $i \in S$ and $u \in U(i)$ when the optimiser is concurrently run with the p-learner is defined as $\widetilde{v}_{t+1}(i,u)$. If this method is to be used when the optimiser is run own $\widetilde{v}_{t+1}(i,u)$ should be substituted for $\widehat{v}_{t+1}(i,u)$.

In Method 3 when in state $i_t = i$ at time t we choose an action $u_t = u$ according to the probability distribution,

$$\frac{\exp\left\{-\gamma'(t)\omega_t(i,u)\right\}}{\sum_{b\in U(i)}\exp\left\{-\gamma'(t)\omega_t(i,b)\right\}} \quad \text{for all} \quad u \in U(i). \tag{3.3.1}$$

The quantity $\omega_t(i,u)$ depends on the magnitude of the current value function estimate at time t defined for the current state i and each possible action $b \in U(i)$.

It measures the relative attraction of taking various actions $b \in U(i)$ in state $i \in S$ at time t. It is formally defined as,

$$\omega_t(i,b) = \frac{\widetilde{v}_{t+1}(i,b) - \min_{u \in U(i)} \{\widetilde{v}_{t+1}(i,u)\}}{\max_{u \in U(i)} \{\widetilde{v}_{t+1}(i,u)\} - \min_{u \in U(i)} \{\widetilde{v}_{t+1}(i,u)\}}. \tag{3.3.2}$$

Initially the quantities $\omega_t(i,b)$ for all $b \in U(i)$ are equal then over time as the algorithm gains more information about the system the quantities start to differ, until eventually if action $u = \arg\min_{u \in U(i)}\{\widetilde{v}_t(i,u)\}$ then $w_t(i,u) = 0$ and if action $u = \arg\max_{u \in U(i)}\{\widetilde{v}_t(i,u)\}$ then $w_t(i,u) = 1$. Thus as t increases if $\gamma'(t)$ tends to a large positive value, eventually the current action estimate $\widetilde{\pi}_{t+1}(i)$ will be the most desirable action in state $i \in S$.

The function $\gamma'(t)$ at each time t is defined as,

$$\gamma'(t) \;=\; \Delta\sigma\gamma(t), \tag{3.3.3}$$

where,

$$\gamma(t) \;=\; \max\left\{0, \frac{t-\mu}{\sigma}\right\} + \exp\left\{-\left|\frac{t-\mu}{2\sigma}\right|\right\}. \tag{3.3.4}$$

The function $\gamma'(t)$ is a smooth function which starts off at zero and gradually increases to infinity. As $t \to -\infty$ the function $\gamma'(t)$ tends to 0 and as $t \to +\infty$ the function is asymptotically $t\Delta$. However we only use a finite sequence of it. At first $\gamma'(t)$ is approximately zero and slowly increases until time μ, allowing us the opportunity to sample the whole state and action space, then after time μ, $\gamma'(t)$ increases linearly in t with gradient Δ so we can focus on the most important states and actions. In Equation (3.3.4) above the parameter μ denotes the amount of time we are willing to sample over the whole state and action spaces, Δ denotes the gradient of the asymptote of $\gamma'(t)$ after time μ, and σ denotes the abruptness of change in $\gamma'(t)$ from the regime of uniform sampling to focused sampling. (see Figure 2.5 above). In this thesis we sometimes refer to μ as the "point of discrimination", this is the point where we discriminate between different actions in each state.

If the product of σ and Δ is small then the function $\gamma'(t)$ is close to its asymptote at each time t. As a result, all actions $b \in U(i)$ in state i are chosen with equal probability up to a point near μ, and after μ some actions are discriminated against. However, as Δ is small we discriminate less than if Δ is large.

If the product of σ and Δ is large, caused by a large magnitude in σ, uniform sampling is not be achieved for very long since $\gamma'(t)$ moves further away from its asymptote at a faster rate than if the product is small. Consequently, the transition between uniform sampling and focused sampling is delayed since the bigger the value of σ the slower the switch between the regime of uniform sampling and focused sampling. An illustration and more formal definitions of parameters μ, σ and Δ can be found Section 2.3.2.3.

3.4 Why we are going to focus on Method 3

In Section 2.3.2 we looked at the three various methods we might implement for choosing actions both in the optimiser and the p-learner. All three have both advantages and disadvantages, but as explained in Section 2.3.2 the disadvantages in Methods 1 and 2 are more apparent. In this section we illustrate the disadvantages of Methods 1 and 2 and show that the results reported in Section 3.10 for Method 3 are far superior. This is the sole reason why, in this chapter, we focus on Method 3 instead of the other two methods. We must state that Method 3 incorporates many of the advantages of Methods 1 and 2. However in order to compare these methods, we had to cover exhaustively what we thought was the best choice of learning parameter sets in each case.

In each method there is a trade-off between the choice of parameters used. For instance, if we used a small T and we focused on learning in the right way then the method's performance could be better, than if we used a large T where the learning is not appropiately focused. This is because with a small T there is a short amount of time in which to sample the true state transition probabilities

corresponding to each state $i \in S$ and each action $\pi^*(i)$. Thus with this in mind it is possible to show that, owing to the discrepancies in the current state transition probabilities compared with the true model corresponding to each state $i \in S$ and each action $\pi^*(i)$, the current value function and policy estimates are closer to the true values for small T than for large T. There is a obvious trade-off between focusing on the right things in each case and learning what the right things are to focus on. We used a large T so we had enough time to sample the model corresponding to each state $i \in S$ and action $\pi^*(i)$, since at the end of the day we want our current solutions to converge to the true solutions. We do this by trying to focus on the state transition probabilities corresponding to each state $i \in S$ and action $\pi^*(i)$ as much as we can, rather than relying on random error to gain good estimates.

In Section 3.3.4 we stated that the reason we do not try to learn about the overall model is because we will not have enough time in which to learn about the important states and actions. We want the algorithms to learn about the overall model to get a good idea of what the optimal policy is. Once the algorithm decides it has the optimal policy, it uses it to focus on the true optimal value function. The problem with Method 1 is that it discriminates between actions before it has a good idea of what the optimal policy is, regardless of the learning parameter sets used. However, an advantage of Method 1 is that, given certain learning parameters, it can sample greedy actions with probability one. An advantage of Method 2 is that it can sample actions with equal probability, to get a good idea of what the optimal policy is. A bad feature of the method is that once it finds the optimal policy, it has difficulties focusing in on the greedy actions with certainty - unless one of its parameters Δ (defined below) is set to infinity. Method 3 on the other hand has the disadvantage in that, given certain learning parameters, it can discriminate between actions far too soon. However, given a good learning parameter set it can both learn about the overall model to get a good idea of what the optimal policy is, and it can focus on greedy actions with

probability either equal to or approaching one.

We ran the optimiser concurrently with the p-learner on the simple discounted cost minimisation problem, described in Section 3.2, using all three methods. Figure 3.1 below illustrates the typical problems the p-learner encountered sampling actions in particular states for Methods 1 and 2. The figure displays the action probabilities for the "hard state", state 4, on the left of the picture and the "very difficult state", state 8, on the right. The top two plots in Figure 3.1 denote the action probabilities for states 4 and 8 using Method 1 with the learning parameter set $\{W = 0.001, M = 1.0\}$, the middle two plots denote the action probabilities for states 4 and 8 using Method 2 with the learning parameter set $\{\mu = 20000, \sigma = 400, \Delta = 1.0\}$, and finally the bottom two plots denote the action probabilities for states 4 and 8 using Method 2 with the learning parameter set $\{\mu = 20000, \sigma = 400, \Delta = 10^5\}$. We compare these results with that of Figures 3.6 and 3.7 to exemplify the advantages of Method 3 (using the learning parameter set $\{\mu = 20000, \sigma = 400, \Delta = 0.01\}$) over the other two methods. The above learning parameter sets were chosen to illustrate the differences in the three methods, because qualitatively speaking they display roughly the same typical behaviour as the other learning parameter sets considered in Section 3.5.1 below.

We recall in Method 1, M defines the asymptotic probability of choosing the current action estimate in any state and W governs the rate at which the probability of choosing the current action estimate in any state converges to M. We chose $M = 1.0$ so that the method chose greedy actions with probability one. In addition we chose $W = 0.001$ because since the method discriminates between actions far too early, it may be a good idea to delay focusing on greedy actions for a while to let the method learn about the overall model. On the other hand, we recall in Method 2 that μ controls the amount of learning we are willing to do, σ controls how quickly we move from uniform sampling to focused sampling, and Δ controls the rate at which we discriminate between action $u \in U(i)$ in state $i \in S$

after time μ. We chose $\mu = 20000$ to let the method do enough sampling and consequently get a good idea of what the optimal policy is. We chose $\sigma = 400$ so that the switch from uniform sampling to focused sampling would not be too slow. Finally we chose $\Delta = 1.0$ and 10^5 to show the difference in behaviour between the two.

Figure 3.1 shows that Method 1 discriminates far too early. Even the currently second best action estimate was discriminated against too soon. In fact this is also true of any learning parameter set considered using Method 1. In state 8 we can see that the p-learner was indecisive about which action to choose. Fortunately on this occasion the p-learner eventually chose the right action, but as a consequence the current action policy did not settle down to the optimal policy until after 29607 iterations. Thus it had less time in which to concentrate its attention on finding the true optimal value function, as explained in the previous section.

As for Method 2, we can see that the method had difficulties focusing on the current action estimates which on this instance were the true optimal actions. The method spent too much time sampling the whole of the state space rather than concentrating its effort on the important states and actions, and eventually sampling each current action with probability one. The method even had difficulties sampling the true optimal actions in the "easy states" with probability one. However, when we increased Δ from 1 to 10^5 the p-learner focused on the true optimal actions in every state, but Figure 3.1 shows that the learning parameters μ and σ cannot be interpreted. For example, the total time allocated for exploration, given a small value of $\sigma = 400$, was 20000 iterations not 14000 iterations as shown in Figure 3.1. The current policy estimates, using both parameter sets, settled down to the true optimal policy after 8909 iterations. We will see later on that this was the same result as Method 3, using any learning parameter set considered in Section 3.5.1 below corresponding to $\mu = 20000$.

When we ran the optimiser concurrently with the p-learner using Method 3 we can see from Figures 3.6 and 3.7 that each optimal action was focused on

with either a probability approaching one or equal to one. We note that in this method, unlike method 2, the values of the learning parameters served their purpose. Other learning parameter sets were also used, and state 8 was the only state that occasionally focused on the suboptimal action rather than the optimal one. However, we must state that this only happened when the point of discrimination came too soon (see Section 3.10), as is almost always the case using Method 1. If we dedicate total learning for too long a time, irrespective of any of these three methods used, good estimates of the transition probabilities corresponding to state $i \in S$ and $\pi^*(i)$ would not be achieved. Hence the current solution estimates would be far from their true values.

In conclusion, Method 3 seems overall to be the best choice of method. Even in this method, however, as we will see later, if not enough learning is done the method will sometimes focus on a suboptimal action in a state where the values of the Q-values are close. Nevertheless, with the learning parameters we chose, the method will always focus on the greedy action with a probability either approaching or equal to one.

3.4 Why we are going to focus on Method 3

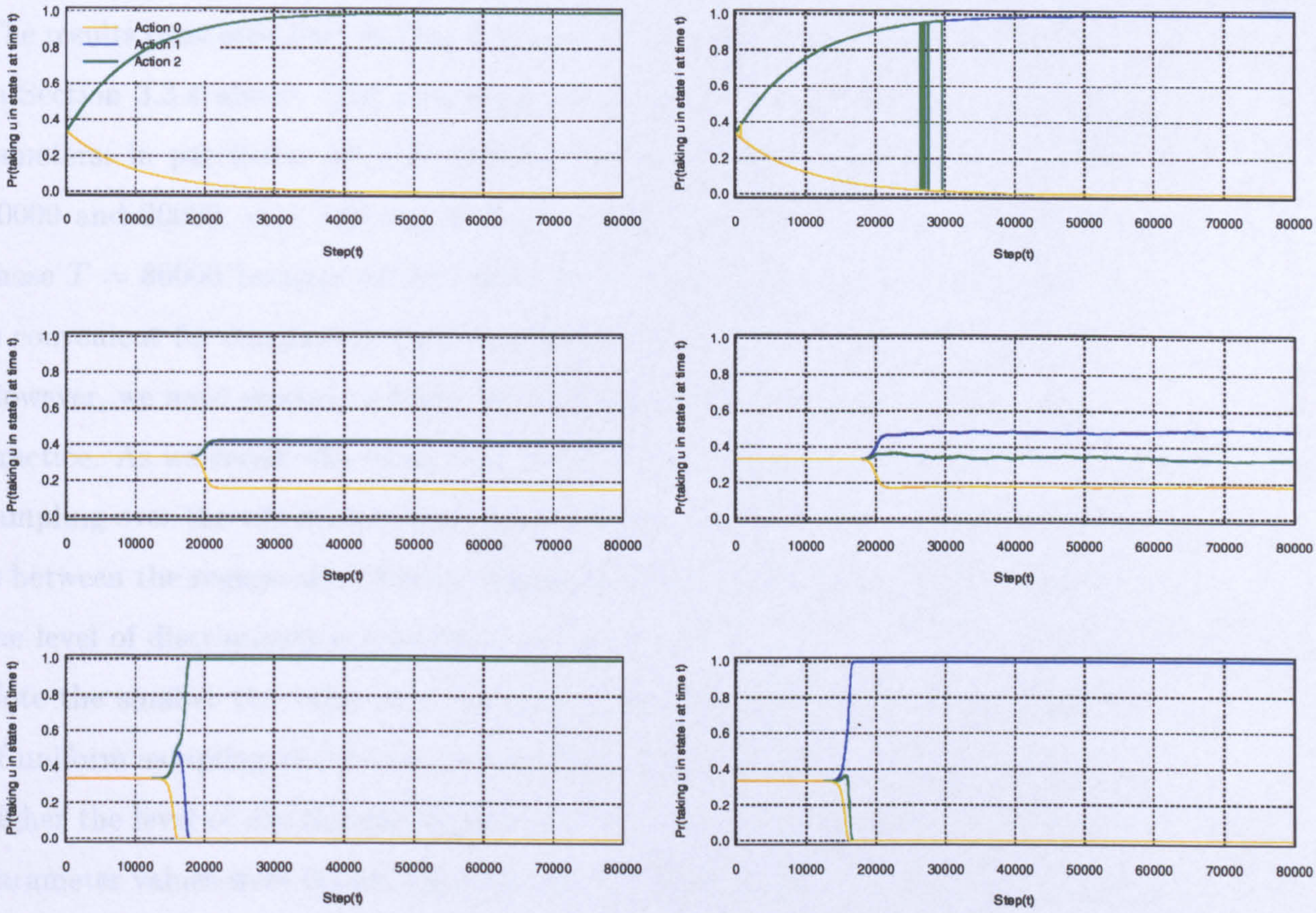


Figure 3.1: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted against time t. From left to right, the top two plots represent the results gained for states 4 and 8 using Method 1 with the learning parameter set $\{W = 0.001,\, M = 1.0\}$, the middle two plots represent the results for states 4 and 8 using Method 2 with the learning parameter set $\{\mu = 20000,\, \sigma = 400,\, \Delta = 1.0\}$, and finally the bottom two plots represent the results for states 4 and 8 using Method 2 with the learning parameter set $\{\mu = 20000,\, \sigma = 400,\, \Delta = 10^5\}$.

3.5 Simulation Procedure

3.5.1 Choice of Parameters

The results presented later in this chapter chose actions using method 3 described in Section 3.3.4 above. The procedures were run using a range of learning parameters; in particular we will describe the results for parameters, $\mu = 5000$, 10000 and 20000, $\sigma = 100$ and 400, $\Delta = 0.003$ and 0.01 and $T = 80000$. We chose $T = 80000$ because we are interested in long runs. Using such a large T is convenient for comparison purposes for other methods throughout this thesis. However, we must stress it is highly unlikely that such long runs would be used in practice. As we recall, the parameter μ denotes the amount of time allocated to sampling over the whole state and action spaces, σ denotes how rapid the switch is between the regime of uniform sampling and focused sampling, and Δ denotes the level of discrimination between actions $u \in U(i)$ in state $i \in S$ after time μ. Note the smaller the value of σ the more rapid the switch between the regime of uniform sampling and focused sampling. Also, the bigger the value of Δ the higher the level of discrimination between actions $u \in U(i)$ in state $i \in S$. These parameter values were chosen because, out of all the various combinations of values we experimented with, they seemed to give the broadest range of behaviour in our learning method. Not only that, they were also the best choice of learning parameters for Method 3 as discussed in Section 3.4. For $\mu = 5000$ we added an extra σ equal to 1000 because if we are restricted to such a small μ we may need to delay the switch from total sampling to focused sampling.

Later on, we will also see how this range of learning parameter values characterise the system and the effect it has on our estimates of $v^*(\cdot)$ and $\pi^*(\cdot)$.

3.5.2 Output

In the optimiser the product at each iteration t are the current value function estimate and the current policy estimate, whereas in the p-learner the products at each iteration t are the action probabilities defined in Equation (2.3.10) using Method 3 and the current state transition probability estimates.

3.5.3 Evaluating Performance

3.5.3.1 Simulations

In some instances of this simple minimisation problem we are interested in how the process evolves along a single run. This can be done using a single seed and should not vary qualitatively from run to run. For other instances we are interested in the variability across runs, done by using several runs with different seeds. We use the single run to see how the range of learning parameters used in our simulations characterise the system, and we use several runs to see the effect the different seeds using different parameter sets have on the estimates of $v^*(\cdot)$ and $\pi^*(\cdot)$ at time T. The two involve different analysis and different output. It should be noted throughout this chapter, twenty seeds were used in total labelled from 1 to 20. The single run consists of results taken from seed 1. We first analyse the results obtained by running the optimiser on its own and then analyse the results obtained when the optimiser is concurrently run with the p-learner.

3.5.3.2 Running the optimiser on its own

In Section 3.7 we present results obtained when running the optimiser on its own in order to establish how well the method will cope with estimating the true optimal value functions $v^*(i)$ and the true optimal actions $\pi^*(i)$ defined for each state $i \in S$. These results act as a bench mark for presenting results when the optimiser is run concurrently with the p-learner. We use various ways of

comparing our estimates with their corresponding true values. Not only that, we compare the speed of convergence of these estimates with that of the Pre-Jacobi.

3.5.3.3 Running the optimiser concurrently with the p-learner

When the optimiser is run concurrently with the p-learner we not only have to compare the current value function estimate $\widetilde{v}_t(\cdot)$ and the current policy estimate $\widetilde{\pi}_t(\cdot)$ at each time t against their corresponding true values, but we have to take in to consideration the current estimates of the true state transition probabilities $\hat{p}_{ij}^t(u)$ because the optimiser's ability to approximate $v^*(\cdot)$ and $\pi^*(\cdot)$ depends on them. Three convergence criteria are investigated in this chapter; the convergence of the current policy estimate, the current state transition probability estimates and the current value function estimate. These are collectively used to consider the trade off between learning and optimising for different learning parameters.

The first of the convergence criteria, the convergence of the current policy estimate, is divided into three sections: Sections 3.8, 3.9 and 3.10. The motivation behind these sections is discussed below.

The algorithms need to have good estimates of the true state transition probabilities and the true value functions for each state-action pair to identify the correct optimal policy; then having identified the optimal policy they need to focus on the state transition probabilities under this policy to identify its correct value function. The purpose of the p-learner is to choose actions so that the resulting transitions enable us to estimate the true state transition probabilities, and the purpose of the optimiser is to visit states to update the current value function estimate and the current policy estimate. Therefore as t increases we learn more about the system and we envisage that the current action estimates $\widetilde{\pi}_t(i)$ will converge to the true optimal actions $\pi^*(i)$ in each state $i \in S$. Thus sampling states and actions following an optimal policy is very important.

Section 3.8 evaluates the behaviour of the system in terms of the proportion

of time spent sampling actions in each state and the proportion of time spent visiting and observing each state in the optimiser and the p-learner respectively. Observing the behaviour of the system in this way is a particular form of looking at the current value function estimate and the current policy estimate and it helps us to identify the effect the different learning parameter sets have on the behaviour of the system. We discovered that the p-learner using a particular learning set focused on the best suboptimal action in one state instead of the optimal action. This motivated further investigation to find out how many times this happened for all the different learning parameters and seeds.

Having characterised the behaviour of the system using the different combinations of parameters, in Section 3.9 we compare how the current policy estimate at each time t differed compared with $\pi^*(\cdot)$. We also check to see if the optimiser's current policy estimates settled down to the optimal policy at roughly the same time as they do when the optimiser is run on its own.

In Section 3.10 further investigation is done by plotting the action probabilities defined in Equation (3.3.1) above. The action probabilities specify the probability of choosing action $u \in U(i)$ in state $i \in S$ at time t. They are important because they determine which actions we choose and hence which state transition probabilities we update in the p-learner. The action probabilities show that we choose the current action estimate with increasing probability and the others with corresponding decreasing (but non zero) probability allowing us to maintain some learning across a diverse range of actions while focusing on what we estimate to be the true actions. We plot the action probabilities for each state using a chosen set of learning parameters. We illustrate how the different sets of learning parameters effect the behaviour of the system and we compare the results to the evolution of the current policy estimates presented in Section 3.9.

We then move on to the second of the convergence criteria, the convergence of the current state transition estimates. This is discussed in Sections 3.11 and 3.12 together with the third convergence criteria, the convergence of the current value

function estimate. The motivation behind presenting the two simultaneously is discussed below.

The current value function estimates depend on the current state transition probability estimates and it is for this reason we present the results for the two simultaneously. In order to compare the performance of different learning parameter sets we had to select a means of measurement. The qualitative performance measure we chose was to look at the root mean square error of the current value function estimates for all $i \in S$ against the root mean square error of the current model estimates, parameterised by time t. This criteria is a good measure of performance to use because the graphs plot the quality of estimating the true optimal value function against the true state transition probabilities for different values of t. We will see the more we learn about the true state transition estimates following an optimal policy, the greater the decrease in the error in the current value function estimate.

In Section 3.11 we compare errors in the current value function estimates with the errors in the current model estimates for different learning parameters. In Section 3.11.4 we explain the phenomena behind the consistent trend in the quality of estimating the optimal value function against the quality of estimating the true state transition probabilities using the results reported in Section 3.9. Section 3.11.5 illustrates the cost of optimising by means of estimating the true state transition probabilities by comparing the quality of estimating the optimal value function over time with two other methods, namely the optimiser when run on its own and the Pre-Jacobi method.

Finally in Section 3.12 we plot the errors in the current value function estimates for all $i \in S$ against the errors in the current model at time T for multiple seeds, for each set of learning parameters, to see if the results shown using the two seeds in Section 3.11 are representative realisations of the process.

We end our analysis of the third convergence criterion in Section 3.13, by summarising the differences in each learning parameter set of the average (multiple

seeds) standard deviation of $\widetilde{v}_T(i)$ plotted against the average bias of $\widetilde{v}_T(i)$. These plots are very informative because they show the average difference in the value function estimates at time T for each state $i \in S$. Section 3.13 uses the analysis of the previous sections in this chapter to explain why certain states have large average biases and large average standard deviations and others do not.

3.6 Summary of Chapter

In this chapter we apply the methodology outlined in Section 3.3 to the simple discounted cost minimisation problem described in Section 3.2 below. The two most important questions we want to address are: (1) relative to the accuracy of the estimates obtained, how much slower or faster is the optimiser compared to standard DP methods? and (2) for a given accuracy, what is the computational cost of not knowing the true state transition probabilities ? This section gives a brief summary of the results in the sections that follow.

In Section 3.7 we show that when we know the true transition probabilities, the current value function estimates and current policy estimates equalled $v^*(i)$ and $\pi^*(i)$ respectively in each state $i \in S$ to 6 decimal places. The same was true for the standard Pre-Jacobi method. In addition, it was found that even though the optimiser is only updated sequentially, it is still as fast at generating optimal solutions.

From Section 3.8 onwards we report the results when the true state transition probabilities are unknown. We first look at the first of the convergence criteria to characterise the behaviour of the system discussed in Section 3.5.3.3. In Section 3.8 we show that sampling actions in different states is highly dependent on the different learning parameters employed. This in turn has a great effect on the ultimate estimates of $v^*(\cdot)$ and $\pi^*(\cdot)$. We show that if we discriminate between actions too soon in the "very difficult" state (state 8), the p-learner sometimes biases on taking a suboptimal action over the optimal one. Section 3.9 shows

that this has a major effect on the greedy policies chosen for different parameters. Also, because of the stochasticity owing to the way we choose actions the time intervals at which the current policy estimate converges to the optimal policy, vary tremendously for different runs. This was not the case when we knew the state transition probabilities. In Section 3.10 we plot the action probabilities which helps us explain the results reported in Sections 3.8 and 3.9. We show that the time intervals at which the current policy estimate oscillated between a suboptimal policy and an optimal policy, follows a similar pattern as the action probabilities plotted after time μ in state 8 in the p-learner.

In Section 3.11 we simultaneously analyse the second and third convergence criteria, previously in discussed in Section 3.5.3.3. The criterion looks at the *quality of the state transition probabilities estimates against the quality of the* true optimal value function estimates. We show that the computational cost of learning about the true state transition probabilities is initially fairly high, but the current value function estimate eventually converges to the true optimal value function. However, the results in this instance are not as accurate as when the true system parameters are known. In Section 3.11.5 we compare the performance of our stochastic methods with that of the Pre-Jacobi method. We show that the root mean square error of the current value function estimate plotted against *time converges to zero in the algorithms where the true system parameters are* known, but not when they are unknown. In Section 3.12 we plotted the quality of the state transition probabilities estimates against the quality of the true optimal value function estimates at time T (the end points) for 20 different runs for each parameter set. It was discovered that there was a noticable change in the quality of learning about the model for different parameter sets, but not in the final estimates of the optimal value function. This makes us wonder whether noise in the current value function estimates will always be present, irrespective of the learning parameter sets used.

Finally in Section 3.13 we plotted the average bias of the current value function estimates $\widetilde{v}_T(i)$ against the average standard deviation of the current value function estimates $\widetilde{v}_T(i)$, for each state $i \in S$. Results were taken from each parameter set considered in this experiment and each observation was averaged over 20 runs. We show that the principal feature in these plots was state 8. This was because for some learning parameter sets the p-learner sampled a suboptimal more often than the optimal action in this particular state. This result was less apparent when we looked at the end points alone.

3.7 Running the optimiser on its own - Convergence of the current value function and current policy estimates

In this section we present results obtained by running the optimiser on its own to see how the method will cope with estimating the true optimal value $v^*(\cdot)$ and the true optimal policy $\pi^*(\cdot)$, as discussed in Section 3.5.3.2 above. The values of the learning parameters considered in this section using Method 3 are $\mu = 5000, 10000, 20000$, $\sigma = 100$ and 400, and $\Delta = 0.003$ and 0.01. An extra σ equal to 1000 is also added for $\mu = 5000$ (see Section 3.5.1). We recall that μ denotes the amount of learning we are willing to do, σ denotes how quickly we move from uniform sampling to focused sampling and Δ denotes the rate at which we discriminate between action $u \in U(i)$ in state $i \in S$ after time μ. Both single and multiple seeds are used depending upon the required output (see Section 3.5.3.1).

In Table 3.3 above we report that the optimiser's approximation of the optimal value function and the optimal policy defined for each state $i \in S$ converged to the true solutions $v^*(i)$ and $\pi^*(i)$, the true values gained solving the problem using the Pre-Jacobi method. This result was true of all parameters and seeds

3.7 Running the optimiser on its own - Convergence of the current value function and current policy estimates

Method	State $i \in S$	$v^*(i) = \widehat{v}_T(i)$	$\pi^*(i) = \widehat{\pi}_T(i)$
Pre-Jacobi	0	1341.166	2
Opt Only		1341.166	2
Pre-Jacobi	1	1318.535	2
Opt Only		1318.535	2
Pre-Jacobi	2	1229.388	2
Opt Only		1229.388	2
Pre-Jacobi	3	1230.372	2
Opt Only		1230.372	2
Pre-Jacobi	4	1254.341	2
Opt Only		1254.341	2
Pre-Jacobi	5	1242.126	2
Opt Only		1242.126	2
Pre-Jacobi	6	1212.976	2
Opt Only		1212.976	2
Pre-Jacobi	7	1342.630	2
Opt Only		1342.630	2
Pre-Jacobi	8	1225.870	1
Opt Only		1225.870	1
Pre-Jacobi	9	1213.414	2
Opt Only		1213.414	2

Table 3.3: The table shows the current value function estimates and the current action estimates for all $i \in S$ found using the Pre-Jacobi method and when the general optimiser was run on its own (Opt Only) after 80000 iterations (fixed simulation horizon T). This was true for all combinations of parameters considered.

considered. The optimiser's approximations of the true optimal value function, $\widehat{v}_T(i)$ for all $i \in S$ were the same as the Pre-Jacobi's method to 10 decimal places.

To investigate which of the above two method's value function estimates converged the fastest (the Pre-Jacobi method and the optimiser when run on its own), we plotted their value function estimates over time from $t = 0$ to 5000 where time t is comparable in both cases. We only plotted the value function estimates from $t = 0$ to 5000 because the estimates using both methods converge fairly quickly. Figure 3.2 below shows that the solutions of the Pre-Jacobi method

converged the fastest using parameters $\mu = 20000.0$, $\sigma = 400.0$, $\Delta = 0.01$. This was true of all parameter sets and seeds considered. We only plotted the results for the current value function estimates at state 1 because qualitatively speaking it displays roughly the same typical behaviour as the other states $i \in S$. After 211 and 2257 iterations, for the Pre-Jacobi method and the optimiser when run on its own respectively, the current value function estimates converged to optimal value functions for each state $i \in S$ to 6 decimal places. We expected this result because the current value function estimates in the Pre-Jacobi method are updated in every state at each iteration whereas the current value function estimates when the optimiser is run on its own are only updated at one state at each iteration.

To investigate the performance of the two methods further, we plotted both curves for step time t' from $t' = 0$ to 2500. In Figure 3.3 t' = time $t/|S|$ for the Pre-Jacobi method and t' = time t for the optimiser when run on its own. This graph gives us a better comparison of the their perfomances as the same amount of work, in each method, is done at each time t'. In Figure 3.3 we plotted the performance of optimiser on its own every time the current value function estimate was updated at state 1, whereas we plotted the performance of the Pre-Jacobi method at every $|S|$ step time t. From Figure 3.3 we can see that the optimiser on its own did very well. However, this was to be expected. Contrary to the Pre-Jacobi method where each state is visited at each time step t, the optimiser is an intelligent algorithm and finds out from experience if the current value estimate at a particular state is worth updating.

3.7 Running the optimiser on its own - Convergence of the current value function and current policy estimates

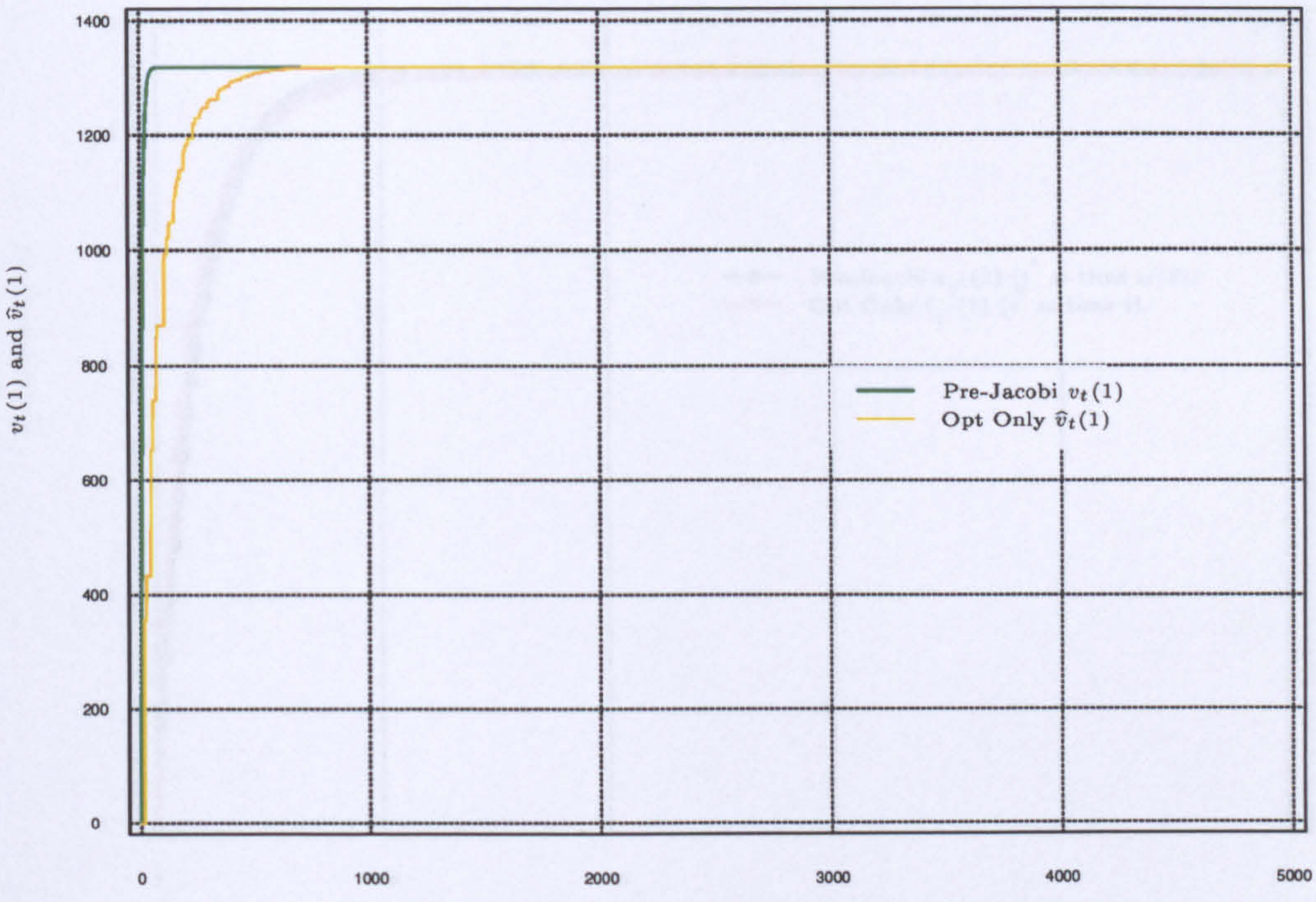


Figure 3.2: The graph above is a plot of the value function estimates for state 1 against step time t (from $t = 0$ to 5000) using parameters $\mu = 20000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ for seed 1 when the optimiser was run on its own (Opt Only) and the Pre-Jacobi (Pre-Jacobi) method. The plot illustrates that the solutions of the Pre-Jacobi ($v_t(1)$) method converge faster than for those when the optimiser is run on its own ($\hat{v}_t(1)$). We plotted the value function estimates for state 1 because qualitatively speaking it displays roughly the same typical behaviour as the other states $i \in S$.

3.7 Running the optimiser on its own - Convergence of the current value function and current policy estimates

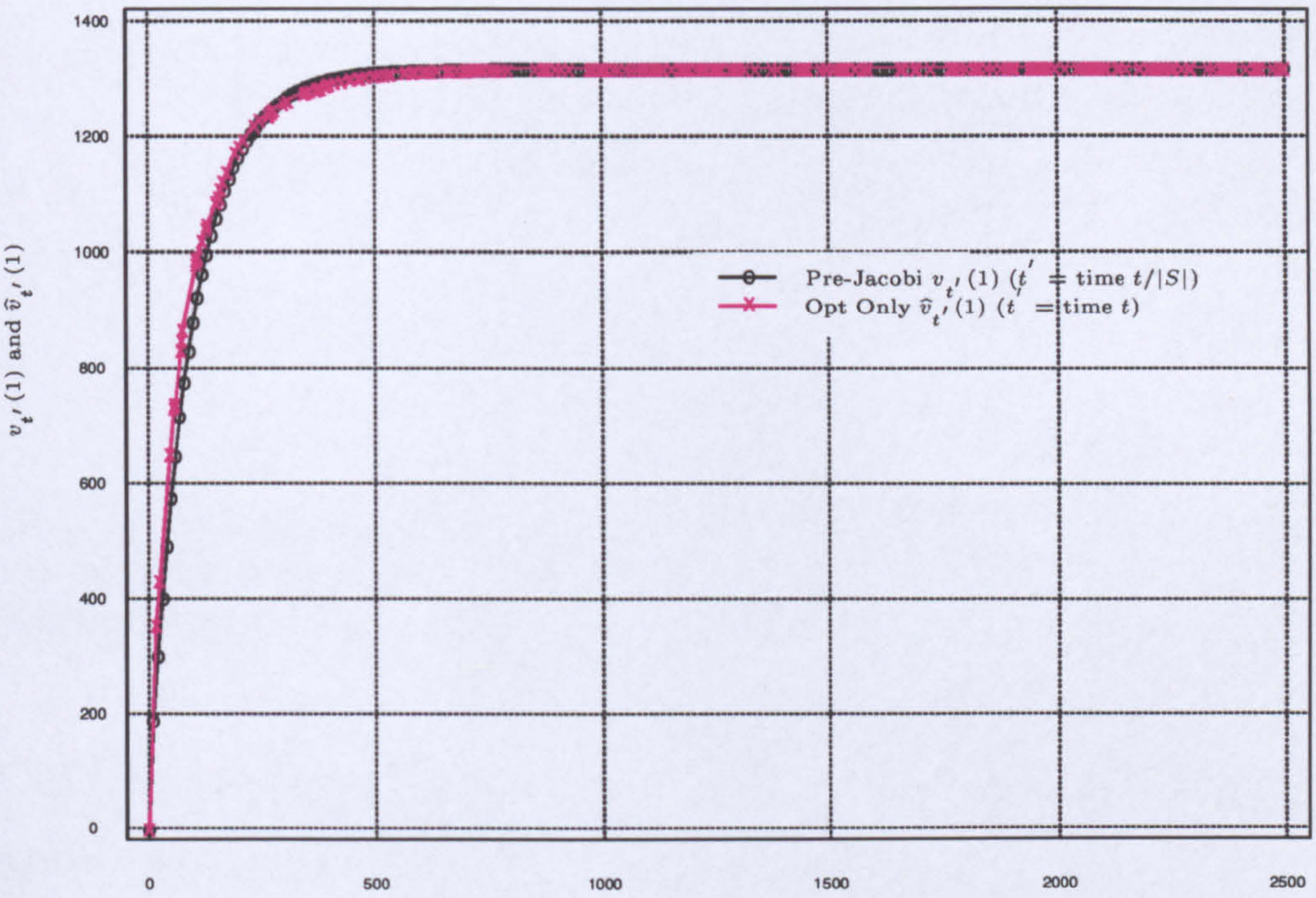


Figure 3.3: The graph above is a plot of the value function estimates for state 1 against step time t' (from $t' = 0$ to 2500) using parameters $\mu = 20000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ for seed 1 when the optimiser was run on its own (Opt Only) and the Pre-Jacobi (Pre-Jacobi) method. Note t' = time $t/|S|$ for the Pre-Jacobi method and t' = time t for the optimiser when run on its own. This graph compares the performance of each method given the same amount of work is done at each time t'. The plot illustrates that the solutions of the optimiser run on its own ($\hat{v}_{t'}(1)$) converges comparably to that of the Pre-Jacobi ($v_{t'}(1)$) method. We plotted the value function estimates for state 1 because qualitatively speaking it displays roughly the same typical behaviour as the other states $i \in S$.

3.7 Running the optimiser on its own - Convergence of the current value function and current policy estimates

We then looked at the time intervals where the current policy estimate $\widehat{v}_t(\cdot)$ equalled the true optimal policy $v^*(\cdot)$ at each time t, as shown in Table 3.4 below.

Method	The time intervals at which $\widehat{\pi}_t(i) = \pi^*(i)$ for all $i \in S$
Opt Only	16 - 21 49 - 50 54 - 80 82 - 92 112 - 135 173 - 80000

Table 3.4: The table shows us the time intervals at which the optimiser when run on its own (Opt Only) chose the true optimal actions in all states $i \in S$. This was true for all combinations of learning parameters considered.

Let us define L to be the time step at which the current policy estimate converged to the true optimal policy.

We can see in Table 3.4 above that the policy estimate initially oscillated between the true optimal policy and a suboptimal policy but the policy estimate finally converged to the true optimal policy after 173 iterations i.e. $L = 173$. This was true for all parameter sets using seed 1. However, it should be stated here that the policy estimate using the Pre-Jacobi method took 3 iterations to converge to the true optimal policy i.e. $L = 3$.

We then considered the average value of L over the 20 different simulations using the 20 different seeds to see how robust and consistent our method really is. In all the combinations of parameters for $\mu = 5000, 10000, 20000$, $\sigma = 100, 400$, and $\Delta = 0.003, 0.01$ the average value of L (over the 20 simulations) was 234 time steps. However, for $\mu = 5000$, $\sigma = 1000$ and $\Delta = 0.003, 0.01$ the average value of L (over the 20 simulations) was 230 and 223 time steps respectively. When $\sigma = 1000$, L is smaller because the order of updating the value function was significantly different due to the large increase in σ. We confirm that all current

policy estimates converged to the true optimal at roughly the same time using the different learning parameter sets and seeds.

In order to investigate whether the method had difficulties finding optimal actions in certain states, we looked at the individual time steps where the current action estimates converged to true optimal actions in each state $i \in S$. Results were recorded for all parameter sets using seed 1. They are characteristic of the results gained using any seed. Table 3.5 below tells us that the optimiser found all of the true optimal actions in each individual state with ease apart from states 4, 5 and 8. This was to be expected after looking at the optimal Q-values for this problem reported in Table 3.2 above. In states 4, 5 and 8, the current

State $i \in S$	The time intervals at which $\widehat{\pi}_t(i) = \pi^*(i)$ for each individual state $i \in S$
0	5 - 80000
1	16 - 80000
2	12 - 80000
3	15 - 80000
4	8 - 21 39 - 135 173 - 80000
5	2 - 33 49 - 80000
6	1 - 80000
7	4 - 80000
8	10 - 50 54 - 80 82 - 92 112 - 163 169 - 80000
9	7 - 80000

Table 3.5: The table shows us the time intervals at which the optimiser chose each optimal action $\pi^*(i)$ in each individual state $i \in S$ using seed 1. This was true for all combinations of parameters considered (excluding the parameter sets corresponding to $\mu = 5000$ and $\sigma = 1000$).

action estimates initially oscillated between the best two actions in each case because of the Q values for the best two actions in each case were fairly close. This happened because the optimiser does not have all the information readily available at any one time as in the Pre-Jacobi method. The optimiser has to wait until all the states have been visited often enough before it finds the true optimal value function and the true optimal actions in all of the states. However, in our experience, in problems when all the state transition probabilities are positive, the method found $v^*(\cdot)$ and $\pi^*(\cdot)$ every time.

3.8 Running the optimiser concurrently with the p-learner - Sampling States and Actions

In the remainder of this chapter we discuss various analyses for running the optimiser concurrently with the p-learner. In the following three sections we are going to look at the first of the convergence criteria, the convergence of the current policy estimates, as discussed in Section 3.5.3.3. In this first section we evaluate the behaviour of the system in terms of the proportion of time spent sampling actions in each state and the proportion of time spent visiting and observing each state in the optimiser and the p-learner respectively. It is a new way of looking at the current policy estimate and the current value function estimate (see Section 3.5.3.3). We use these results to identify the effect different learning parameters have on the system. The values of the learning parameters considered in this section are $\mu = 5000, 10000, 20000$, $\sigma = 100$ and 400, and $\Delta = 0.003$ and 0.01 with an extra σ equal to 1000 included for $\mu = 5000$ (see Section 3.5.1). We recall that μ denotes the amount of learning we are willing to do, σ denotes how quickly we move from uniform sampling to focused sampling and Δ denotes the rate at which we discriminate between action $u \in U(i)$ in state $i \in S$ after time μ. The smaller the value of σ the quicker the transition between total sampling

and focussed sampling. Also, the bigger the value of Δ the quicker the rate of discrimination between actions after time μ. Again, both single and multiple runs are used depending upon the required output (see Section 3.5.3.1).

Tables 3.6 and 3.7 below show (for learning parameters $\mu = 10000$, 20000, $\sigma = 400$ and $\Delta = 0.01$) the percentage of time for which we visited each state and the percentage of time for which we observed each state in the optimiser and p-learner respectively, and the percentage of time for which we sampled each action in each particular state in both. These tabulated results typify the differences in behaviour between the optimiser and the p-learner when the same set of learning parameters are used in each. Tables 3.8 (for parameters $\mu = 10000$, $\sigma = 100$ and $\Delta = 0.003, 0.01$) and 3.9 (for parameters $\mu = 5000$, $\sigma = 400, 1000$ and $\Delta = 0.01$) below also report the percentage of time for which we observed each state and the percentage of time for which we sampled each action in each particular state in the p-learner. They, along with Table 3.7, are used to illustrate the effects the different learning parameters have on the p-learner's ability to choose optimal actions in each state. Note, all of the bold figures in the above tables correspond to the true optimal actions $\pi^*(i) \in U(i)$ in each state $i \in S$. Finally, Table 3.10 below illustrates the effect the different seeds have on choosing the current action in state 8, the "very difficult" state.

Although the function of Table 3.6 is to enable us to compare the proportion of time spent visiting each state in the optimiser with the proportion of time spent visiting each state in the p-learner, shown in Table 3.7, we also note that the proportion of time spent sampling current action estimates in each state $i \in S$ is comparable in both tables. This was true for all parameter sets considered in this sequence of simulations. However it is not suprising since in each simulation we used the same values for μ, σ Δ in both the optimiser and p-learner, and the only thing that was different was the means of transition between states.

In Table 3.6 we can see the number of visits across each state in the optimiser is more evenly spread compared with the states observed in the p-learner shown

	$\mu = 10000\ \sigma = 400\ \Delta = 0.01$				$\mu = 20000\ \sigma = 400\ \Delta = 0.01$			
State $i \in S$	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	2	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	2
0	0.085	0.038	0.047	**0.915**	0.082	0.086	0.095	**0.819**
1	0.093	0.039	0.041	**0.920**	0.090	0.084	0.088	**0.829**
2	0.089	0.043	0.042	**0.915**	0.090	0.090	0.084	**0.826**
3	0.093	0.041	0.050	**0.909**	0.096	0.089	0.092	**0.819**
4	0.105	0.033	0.110	**0.857**	0.107	0.072	0.152	**0.776**
5	0.105	0.036	0.059	**0.905**	0.105	0.077	0.097	**0.827**
6	0.110	0.033	0.035	**0.932**	0.112	0.072	0.070	**0.858**
7	0.100	0.034	0.037	**0.929**	0.097	0.079	0.083	**0.837**
8	0.106	0.032	**0.047**	0.921	0.102	0.074	**0.838**	0.088
9	0.113	0.035	0.037	**0.927**	0.118	0.070	0.072	**0.857**

Table 3.6: The above table represents the percentage of time for which we spent visiting each state and the percentage of time for which we spent in each action for each state in the optimiser. Seed 1 was used to generate the results using $\mu = 10000, 20000$, $\sigma = 400$, and $\Delta = 0.01$. The bold figures represent the true optimal actions in each state.

in Table 3.7. This is due to the temperature schedule used in the optimiser, the sole purpose of which is initially to visit each state with uniform probability and then sample states according to the true state transition probabilities (see Section 2.2.2), whereas the p-learner only visits each state according to the true state transition probabilities (see Section 2.3). This too was true for all parameter sets considered in this experiment.

To observe the effects of varying μ we report the results quoted in Table 3.7 above. In Table 3.7 the results show that for $\mu = 10000$ the p-learner spent most of the time sampling current action estimates. In the "easy states" the p-learner sampled the other actions in each state only 3.3% to 5.4% of the time whereas in the "hard states", states 4 and 5, the other actions were sampled between 3.7% and 11.8% of the time. However, in state 8, the "very difficult state", the p-learner sampled the best suboptimal action 91.4% of the time. Here, we can see

3.8 Running the optimiser concurrently with the p-learner - Sampling States and Actions

	$\mu = 10000\ \sigma = 400\ \Delta = 0.01$				$\mu = 20000\ \sigma = 400\ \Delta = 0.01$			
State $i \in S$	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	2	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	2
0	0.076	0.039	0.039	**0.922**	0.073	0.082	0.088	**0.830**
1	0.088	0.035	0.034	**0.931**	0.083	0.083	0.075	**0.842**
2	0.080	0.042	0.049	**0.909**	0.084	0.084	0.093	**0.823**
3	0.086	0.039	0.054	**0.907**	0.091	0.085	0.099	**0.816**
4	0.110	0.038	0.118	**0.844**	0.113	0.083	0.160	**0.757**
5	0.106	0.037	0.058	**0.905**	0.104	0.078	0.094	**0.828**
6	0.111	0.034	0.034	**0.932**	0.114	0.073	0.067	**0.860**
7	0.102	0.037	0.039	**0.924**	0.098	0.086	0.084	**0.830**
8	0.109	0.033	**0.053**	0.914	0.104	0.077	**0.830**	0.093
9	0.130	0.033	0.034	**0.933**	0.135	0.068	0.067	**0.865**

Table 3.7: The above table represents the percentage of time for which we spent observing each state and the percentage of time for which we spent sampling each action within each state in the p-learner. Seed 1 was used to generate the results using $\mu = 10000, 20000$, $\sigma = 400$, and $\Delta = 0.01$. The bold figures represent the true optimal actions in each state.

the p-learner took the wrong action far too often. An explanation of the reasons why the currently second best action estimate in the "hard states" is sampled more often than the currently second best action estimate in any other state, can be found in the next section.

For $\mu = 20000$ the p-learner spent more time sampling actions other than the current action estimates compared with when we set $\mu = 10000$. This makes sense as we specified that we were willing to sample each action $u \in U(i)$ in state $i \in S$ for twice as long using $\mu = 20000$ compared with $\mu = 10000$. In the "easy states" the p-learner sampled actions other than current action estimates between 6.7% to 9.9% of the time which is roughly twice the amount of time compared with when μ equalled 10000. The same can be said in the "hard states", but in state 8 the p-learner this time sampled the correct action for the majority of the time (83.0%), unlike the $\mu = 10000$ case. It seems that for $\mu = 10000$ we were

forced to make a decision far too early, and delaying the decision using $\mu = 20000$ meant that we bided our time for twice as long but we eventually made the right decision. This is our main concern when it comes to sampling. We note the proportion of time the p-learner spent observing each state is comparable in both cases. This was true for any learning parameter set used in this experiment.

Reducing the value of μ does not necessarily mean that in some states the p-learner will definitely focus on a suboptimal action instead of the optimal one, but it does increase the chances of doing so because the regime of total sampling is somewhat lessened. Out of all the various combinations of parameters we experimented with, the parameter set $\{\mu = 10000, \sigma = 400, \Delta = 0.01\}$ in this single run was the only learning set where any current action estimate settled down to a suboptimal action. In the other cases (using the same values for σ and Δ for both $\mu = 20000$ and 10000 where $\widetilde{\pi}_T(i) = \pi^*(i)$ for each state $i \in S$) each action, other than the current action estimates, was sampled for twice as long for $\mu = 20000$ compared with $\mu = 10000$.

To illustrate the effect of varying σ by a small amount we concentrate on the results gained from the learning parameters $\{\mu = 10000, \sigma = 400, \Delta = 0.01\}$ and compare them with the ones gained from $\{\mu = 10000, \sigma = 100, \Delta = 0.01\}$. The results are displayed in Tables 3.7 and 3.8 respectively. We can see by comparing the two tables that over all for $\sigma = 100$ all the actions other than the current action estimates, where $\widetilde{\pi}_T(i) = \pi^*(i)$ for each state $i \in S$ in both sets, were sampled for a little longer. This was to be expected as the lower the value of σ the more time is spent sampling over each action $u \in U(i)$ in each state $i \in S$, but such a small difference in sampling was caused by only a small decrement in σ. In fact this result was true of any learning set we used in our sequence of simulations where $\widetilde{\pi}_T(i) = \pi^*(i)$ for each state $i \in S$. However, we must state that such a small decrement in σ did enable the p-learner to focus on each optimal action in each state. Later we will investigate whether this happens when σ is increased from $\sigma = 400$ to 1000 using $\mu = 5000$.

3.8 Running the optimiser concurrently with the p-learner - Sampling States and Actions

	$\mu = 10000\ \sigma = 100\ \Delta = 0.003$				$\mu = 10000\ \sigma = 100\ \Delta = 0.01$			
State $i \in S$	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	2	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	2
0	0.070	0.049	0.050	**0.901**	0.073	0.047	0.044	**0.909**
1	0.081	0.045	0.043	**0.913**	0.083	0.042	0.040	**0.918**
2	0.085	0.047	0.055	**0.898**	0.083	0.045	0.054	**0.900**
3	0.091	0.044	0.063	**0.893**	0.090	0.043	0.051	**0.906**
4	0.111	0.046	0.250	**0.705**	0.111	0.044	0.114	**0.841**
5	0.103	0.044	0.100	**0.855**	0.106	0.042	0.057	**0.901**
6	0.118	0.038	0.037	**0.925**	0.116	0.037	0.033	**0.931**
7	0.096	0.047	0.049	**0.904**	0.096	0.044	0.047	**0.909**
8	0.104	0.043	**0.899**	0.058	0.104	0.041	**0.902**	0.056
9	0.140	0.037	0.035	**0.928**	0.138	0.035	0.034	**0.931**

Table 3.8: The above table represents the percentage of time for which we spent observing each state i and the percentage of time for which we spent sampling each action $u \in U(i)$ in the p-learner. Seed 1 was used to generate the results using $\mu = 10000$, $\sigma = 100$ and $\Delta = 0.003, 0.01$. The bold figures represent the true optimal actions in each state.

We then considered the effect that differing the values of Δ had on the behaviour of the system. To compare the effect we report the results gained from the learning parameter sets ($\{\mu = 10000, \sigma = 100, \Delta = 0.003\}$) and ($\{\mu = 10000, \sigma = 100, \Delta = 0.01\}$) shown in Table 3.8 above. The table shows that the current action estimates were sampled for longer when Δ was set to 0.01 compared with 0.003. Thus for $\Delta = 0.01$ the other actions were sampled a little less often. This especially happened in the "easy states" and the "very difficult state". However in the "hard states", states 4 and 5, we note a vast improvement. The second best action was discriminated against more, by a factor of almost 2, when Δ equalled 0.01 compared with when it equalled 0.003. This is because after time μ, $\Delta = 0.01$ caused the p-learner to put more weight in to choosing the current action estimate over the currently second best action estimate compared with $\Delta = 0.003$. Again we report the same happened in all the learning parameter

sets we experimented with. We will see a better illustration of this when we plot the action probabilities in Section 3.10 below.

	$\mu = 5000\ \sigma = 400\ \Delta = 0.01$				$\mu = 5000\ \sigma = 1000\ \Delta = 0.01$			
State $i \in S$	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	2	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	2
0	0.073	0.018	0.018	**0.964**	0.072	0.004	0.011	**0.985**
1	0.084	0.016	0.017	**0.968**	0.084	0.005	0.007	**0.988**
2	0.081	0.022	0.021	**0.957**	0.081	0.006	0.008	**0.986**
3	0.089	0.017	0.027	**0.956**	0.089	0.004	0.020	**0.975**
4	0.109	0.015	0.083	**0.902**	0.110	0.003	0.087	**0.909**
5	0.107	0.016	0.037	**0.947**	0.106	0.004	0.038	**0.958**
6	0.118	0.017	0.016	**0.967**	0.119	0.006	0.008	**0.986**
7	0.096	0.017	0.017	**0.966**	0.095	0.005	0.006	**0.988**
8	0.104	0.014	**0.888**	0.098	0.103	0.004	**0.957**	0.039
9	0.140	0.015	0.014	**0.971**	0.142	0.004	0.007	**0.989**

Table 3.9: The above table represents the percentage of time for which we spent observing each state i and the percentage of time for which we spent sampling each action $u \in U(i)$ in the p-learner. Seed 1 was used to generate the results using $\mu = 5000$, $\sigma = 400, 1000$ and $\Delta = 0.01$. The bold figures represent the true optimal actions in each state.

We then looked at the effect of restricting μ to 5000, to see if this significantly changed the behaviour of the system. Obviously with $\mu = 5000$ the p-learner would be forced to make an early decision as to what is the optimal action in each state. However, making a decision too early may result in serious consequences. We chose to use $\sigma = 1000$. This meant that the p-learner would discriminate between the actions sooner but there would be a delay between total learning and focused sampling, in the hope that if it did make a wrong decision early on it would be given a chance to recover.

We see can see from Table 3.9 above, that the percentage of time the p-learner sampled the current action estimate in each state increased when σ was increased. The p-learner hardly visited any of the other actions in any of the

states other than the current action estimates - which in this case were the true optimal actions in each state $i \in S$. In the "easy states" using $\sigma = 1000$ the p-learner sampled actions other than the current action estimates 1.1% of the time at most and it sampled the currently second best action estimate in state 4 and 5, 8.7% and 3.8% of the time respectively. As for the difficult state, state 8, using $\sigma = 1000$ the p-learner sampled the currently second best action estimate 3.9% of the time. However, when we decreased σ to 400 the results are comparable but the p-learner did sample the true optimal actions in each state a little less often. We should state that the effect of sampling actions in each state was more obvious when we increased σ by such a large amount. Again we note that the proportion of time spent observing each state was comparable in the two cases. Ostensibly the results for $\{\mu = 5000, \sigma = 400, 1000 \text{ and } \Delta = 0.01\}$ look good for seed 1 because they focus on the true optimal actions for longer, than any of the other parameters used in this single run, but they are very misleading.

The choices of μ, σ and Δ interact but not all values of μ will be appropriate for certain values of σ and Δ, and vice versa. We recorded the number of times the p-learner chose the true optimal action in state 8 out of the 20 runs for each parameter set considered in our sequence of simulations. Table 3.10 below tells us that for $\{\mu = 5000, \sigma = 1000 \text{ and } \Delta = 0.01\}$ the p-learner only chose the optimal action 13 times out of the 20 runs. On the whole, the results for $\mu = 5000$ are not as good as the results obtained using $\mu = 10000$ or $\mu = 20000$. This evidence suggests if the p-learner does not do a sufficient amount of learning, the p-learner may be forced to "home-in" on a suboptimal action in state 8 instead of the true optimal one. In fact the p-learner sampled the best suboptimal action in state 8 in the cases where it did not focus on the optimal action.

In conclusion, $\mu = 5000$ would appear to be a bad choice. It does not allow the p-learner to do enough learning because it has to make a decision far too early. The results show when μ is small choosing a large σ is not always necessarily a good option and it may be better to go for a small value of σ to do more

μ	σ	Δ	The no. of times out of the 20 runs for which $\widetilde{\pi}_T(8) = \pi^*(8)$
20000	400	0.01	20
20000	400	0.003	18.5
20000	100	0.01	20
20000	100	0.003	19
10000	400	0.01	16.5
10000	400	0.003	20
10000	100	0.01	20
10000	100	0.003	20
5000	1000	0.01	13
5000	1000	0.003	17
5000	400	0.01	18
5000	400	0.003	17
5000	100	0.01	16
5000	100	0.003	18

Table 3.10: The table above shows us the number of times out of the 20 runs (for each set of parameters) we focused on the true optimal action in state 8, "the very difficult" state when the optimiser was run concurrently with p-learner, for which $\widetilde{\pi}_T(8) = \pi^*(8)$.

learning. Perhaps $\mu = 5000$ is just too small a value for μ. As for $\mu = 10000$ and 20000 for the majority of time the p-learner did do enough learning in all parameter sets used in this study. We report in every simulation the current action estimate converged to the true optimal action in all of the states bar state 8. In the next section we look at how these different learning parameter sets affect the optimiser's ability to focus on the optimal policy.

3.9 Running the optimiser concurrently with the p-learner - Convergence of the current policy estimate

This section is the second of the three sections concerned with looking at the first of the convergence criteria, the convergence of the current policy estimates, as discussed in Section 3.5.3.3. Having characterised the behaviour of the system for different learning parameter sets in the previous section, we are now in a position to look at evaluatory and diagnostic ways of establishing whether our results have converged to their corresponding optimal solutions. We compare these results with the optimal solutions and that of Section 3.7, when the optimiser was run on its own. As with all sections presented in this case study the learning parameter sets $\{\mu, \sigma, \Delta\}$ considered are the ones found using the various combinations of $\mu = 5000, 10000, 20000$, $\sigma = 100$ and 400, and $\Delta = 0.003$ and 0.01 with an extra σ equal to 1000 included for $\mu = 5000$ (see Section 3.5.1). We recall that μ controls the amount of learning, σ controls the transition from total sampling to focused sampling and Δ controls the rate at which we discriminate between action $u \in U(i)$ in state $i \in S$ after time μ. The results in this section consist of both single seeds and multiple seeds depending upon the required output (see Section 3.5.3.1).

Tables 3.11 and 3.12 below illustrate the typical problems encountered by the optimiser when run concurrently with the p-learner. Table 3.11 consists of the time intervals where the current action estimates converged to the true optimal action in each individual state using learning parameter sets corresponding to $\mu = 20000$, whereas Table 3.12 reports the last time intervals where the optimiser's current policy estimates settled down to the true optimal policy using the learning parameter set $\{\mu = 2000, \sigma = 400, \text{and } \Delta = 0.01\}$ for multiple runs. Figures 3.4 and 3.5 below, on the other hand, give us a clearer indication of how the different

sets of learning parameters compared. In Figures 3.4 and 3.5 we display the time intervals where the current policy estimate actually equalled the true optimal policy and those intervals where the current policy estimate was equal to some suboptimal policy. (An optimal policy is a set of decision rules that specify the

State	The time intervals at which $\widetilde{\pi}_t(i) = \pi^*(i)$ for each individual states $i \in S$ the time intervals
0	12 - 80000
1	4 - 80000
2	8 - 80000
3	9 - 80000
4	13 - 80000
5	2 - 80000
6	1 - 80000
7	7 - 80000
8	483 - 1628 1737 - 1747 2277 - 2923 2943 - 2947 2967 - 3059 3599 - 3648 6044 - 6067 6161 - 6210 6247 - 7997 8069 - 8464 8713 - 8885 8908 - 80000
9	3 - 80000

Table 3.11: The table shows us the time intervals at which the optimiser when run concurrently with the p-learner chose an optimal action in each particular state for the combinations of learning parameters $\mu = 20000$, $\sigma = 100, 400$ and $\Delta = 0.003, 0.01$ and $T = 80000$ using seed 1.

correct action in each state; any other set of decision rules is a suboptimal policy). Although the figures do not differentiate between different suboptimal policies,

in general the actual suboptimal policy the optimiser used was the policy where the current action estimates were equal to the true optimal actions in all of the states, except state 8; the state in which the p-learner some times focuses on action 2 instead of action 1.

Table 3.11 shows that the optimiser had trouble differentiating between the true optimal action and a suboptimal action in state 8. This was true for most of the learning parameter sets and seeds considered, but on the whole most of the current policy estimates eventually settled down to the true optimal policy. We predicted problems such as these may occur in certain states when we reported the values of the true optimal "Q-values" in Table 3.2 (Section 3.2). The reason why it happened in state 8 was that the Q-values for the best two actions were extremely close. In fact we also stated that we may expect to see difficulties in states 4 and 5, but in truth it rarely happened. However, if we look back to the previous section we can see that the p-learner spent more time sampling the second best action in states 4 and 5 than in both the "easy" set of states, and the "very difficult " state when $\tilde{\pi}_T(8) = \pi^*(8)$; this is because the Q-values for the two best actions in states 4 and 5 were reasonably close relative to the range of the Q-values in each of these states. In state 8 the p-learner focused only on a suboptimal action or the true optimal action for the most part because the range of the Q-values in state 8 was small (see Table 3.2). We will see this more clearly in the next section when we plot the action probabilities defined in Equation (3.3.1) above. If we compare the results taken from Table 3.11 with the ones taken from Table 3.5, when the optimiser was run on its own, we can see that in all of the states bar state 8 the results are comparable. This was true for all the parameters and seeds considered. However because of the difficulties in focusing on the true optimal action in state 8, the times at which the current policy estimates settled down to the true optimal policy varied tremendously for different seeds when the optimiser was run concurrently with the p-learner. This is illustrated in Table 3.12 below. These results are contrary to those found

3.9 Running the optimiser concurrently with the p-learner - Convergence of the current policy estimate

Seed	The last time intervals at which $\widetilde{\pi}_t(\cdot) = \pi^*(\cdot)$ for different seeds
1	8909 - 80000
2	42293 - 80000
3	25507 - 80000
4	3134 - 80000
5	12513 - 80000
6	1289 - 80000
7	35259 - 80000
8	27966 - 80000
9	4111 - 80000
10	1840 - 80000
11	367 - 80000
12	144 - 80000
13	17908 - 80000
14	17407 - 80000
15	12424 - 80000
16	3850 - 80000
17	3105 - 80000
18	2177 - 80000
19	9694 - 80000
20	597 - 80000

Table 3.12: The table shows us the last time interval where the optimiser's (when run concurrently with the p-learner) current policy estimates converged to the true optimal policy for $\mu = 20000$, $\sigma = 400$, $\Delta = 0.01$, and $T = 80000$ using seeds labelled 1 to 20. This was typical of any parameter set used in our sequence of simulations.

when the optimiser was run on its own (see Section 3.7). This happened because initially the p-learner uses a uniform prior for all the state-transition probabilities and the Q-values in the two best actions in state 8 are extremely close. The only way we can distinguish between the two best actions in each state 8 is by learning about the true state transition probabilities, and this takes time. In the other states, where the overall costs of the best two actions are not so close to each other, this problem does not arise.

From Figure 3.4 we can see that when $\mu = 20000$ the optimiser chose the true

optimal policy in the same intervals for each state irrespective of the values of σ and Δ used (the actual intervals are the ones where $\widetilde{\pi}_T(8) = \pi^*(8)$ in Table 3.11). This is because $\mu = 20000$ using small values of σ and Δ caused the algorithm to sample each action in each state with uniform probability for approximately 20000 iterations (an illustration of this will be shown in the action probability plots in the next section), and by the time we had got to the point of discrimination (time μ) in each case the p-learner had decided which action it was going to focus on in each state. Note the current policy estimate had settled down to the true optimal policy by 8908 iterations. We also note that all the optimal actions were found in all of the states fairly quickly except in state 8, the "very difficult state". This was true for most of the learning parameter sets considered, but in some cases the current action estimates did not converge to the true optimal action in state 8. We will see this in Figures 3.4 and 3.5 below.

Figure 3.4 shows that in all learning parameter sets using $\mu = 10000$ bar one, the current policy estimates settled down to the true optimal policy. In Figure 3.4 we can see that the learning parameter set $\{\mu = 10000, \sigma = 100, \Delta = 0.003\}$ chose the optimal policy in the same intervals as $\mu = 20000$ because these learning parameters too caused $\gamma'(t)$ to be very close to its asymptote. As for the rest of the $\mu = 10000$ sets because the product of σ and Δ was bigger the p-learner discriminated between actions sooner, and consequently uniform sampling was achieved for less. As a result (for this problem and seed at least) the p-learner did not do enough learning, hence the bigger we made the value of $\sigma\Delta$ the more the optimiser seemed to struggle. It is clear from Figure 3.4 that the function $\gamma'(t)$ with the largest product of σ and Δ ($\mu = 10000, \sigma = 400, \Delta = 0.01$) loses the optimal policy after 22032 iterations and never recovers. This corresponds to the results presented in Table 3.7 where the parameter set $\{\mu = 10000, \sigma = 400, \Delta = 0.01\}$ was the only set that focused on the suboptimal action in state 8. In the next section we will look at the action probability plots to explain this phenomena.

3.9 Running the optimiser concurrently with the p-learner - Convergence of the current policy estimate

Figure 3.5 on the other hand illustrates how quickly the current policy estimates converged to the true optimal policy using the different sets of learning parameters for $\mu = 5000$. The figure shows that all the current action policies eventually converged to the true optimal policy. These results again correspond to the ones reported in the previous section. For $\sigma = 100$ and $\Delta = 0.003$ and 0.01 the optimiser chose the optimal policy in the same intervals as the $\mu = 20000$ case up to time 3648, then it sampled actions in a different order because the discrimination point came approximately 15000 iterations sooner. For $\sigma = 400$ and $\Delta = 0.003$ the optimiser chose the optimal policy in the same time intervals as the 20000 case to begin with, then because of the nature of $\gamma'(t)$ (using a smaller value of μ) the p-learner sampled other actions. Consequently, the optimiser's current policy estimates settled down to the true optimal policy at different time intervals. The same can be said for the other sets of learning parameters considered for $\mu = 5000$. We will see a better illustration of this in the next section.

In conclusion if the Q-values of the two best actions were not so close, as in state 8, the problem of continual indecision about which action to focus on would not arise. We have also found that the intervals at which the optimiser's current policy estimates finally settled down to the true optimal policy varied considerably, for different seeds and learning parameters. This too stems from state 8. The problem is not because the optimiser does not update particular states often enough, it is due to stochasticity caused by the way we choose actions using different seeds and learning parameter sets. We have evidence of this below. In the previous section we recorded the percentage of time we spent visiting each state, and the results showed that each state was updated fairly often. Also in Section 3.7 we showed that if we know the true state transition probabilities, irrespective of the learning parameter set and seed used, the current policy estimate converged to the true optimal policy with ease and at roughly the same time;

Table 3.12 clearly shows this does not happen when the optimiser is run concurrently with the p-learner. Therefore the problems we have encountered are not caused by not updating each state often enough, but are caused by the fact that we do not have good enough estimates of the true state transition probabilities.

When we used a set of learning parameters with a large product of σ and Δ we can see it took the optimiser's current policy estimates longer to converge to the true optimal policy, all, that is, except the learning parameter sets corresponding to $\mu = 5000$ and $\sigma = 1000$. (This will not always be the case because it depends on the problem at hand). In the $\mu = 5000$ and $\sigma = 1000$ cases the p-learner was forced to make a decision far too soon but fortunately it made the correct decision. However we have evidence to suggest from Table 3.10 that $\sigma = 1000$ is too big a value to use with $\mu = 5000$, at least in the case of this problem.

3.9 Running the optimiser concurrently with the P-learner - Convergence of the current policy estimate

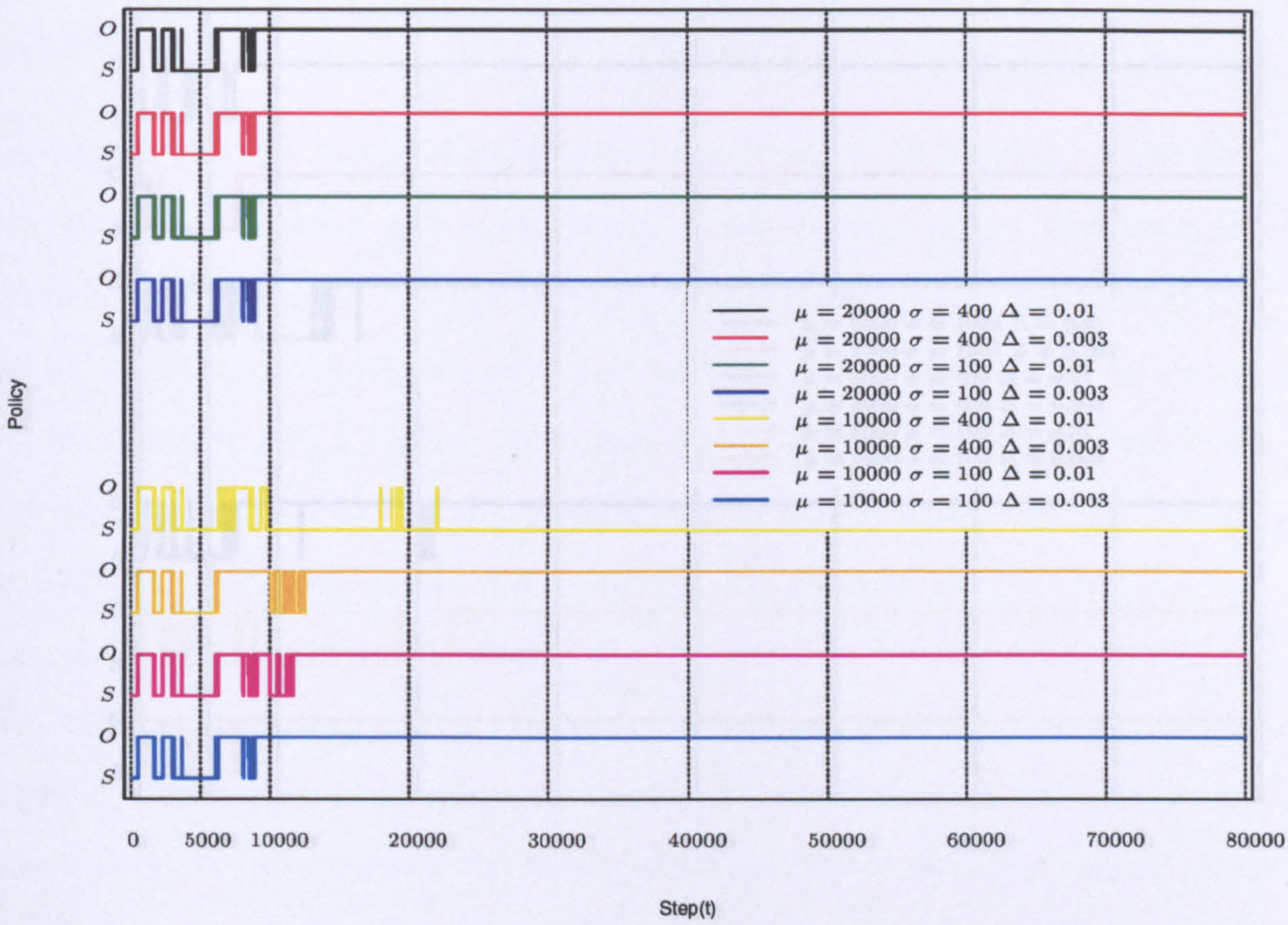


Figure 3.4: The graph above shows us the time intervals when the optimiser's (when concurrently run with the p-learner) current policy estimates converged to the true optimal policy (O) and a suboptimal policy (S) respectively for the different combinations of parameters $\mu = 10000, 20000$, $\sigma = 100, 400$ and $\Delta = 0.003, 0.01$ and $T = 80000$ for seed 1. We can see in all parameter sets the current policy estimate oscillated between the optimal policy and a suboptimal policy.

3.9 Running the optimiser concurrently with the p-learner - Convergence of the current policy estimate

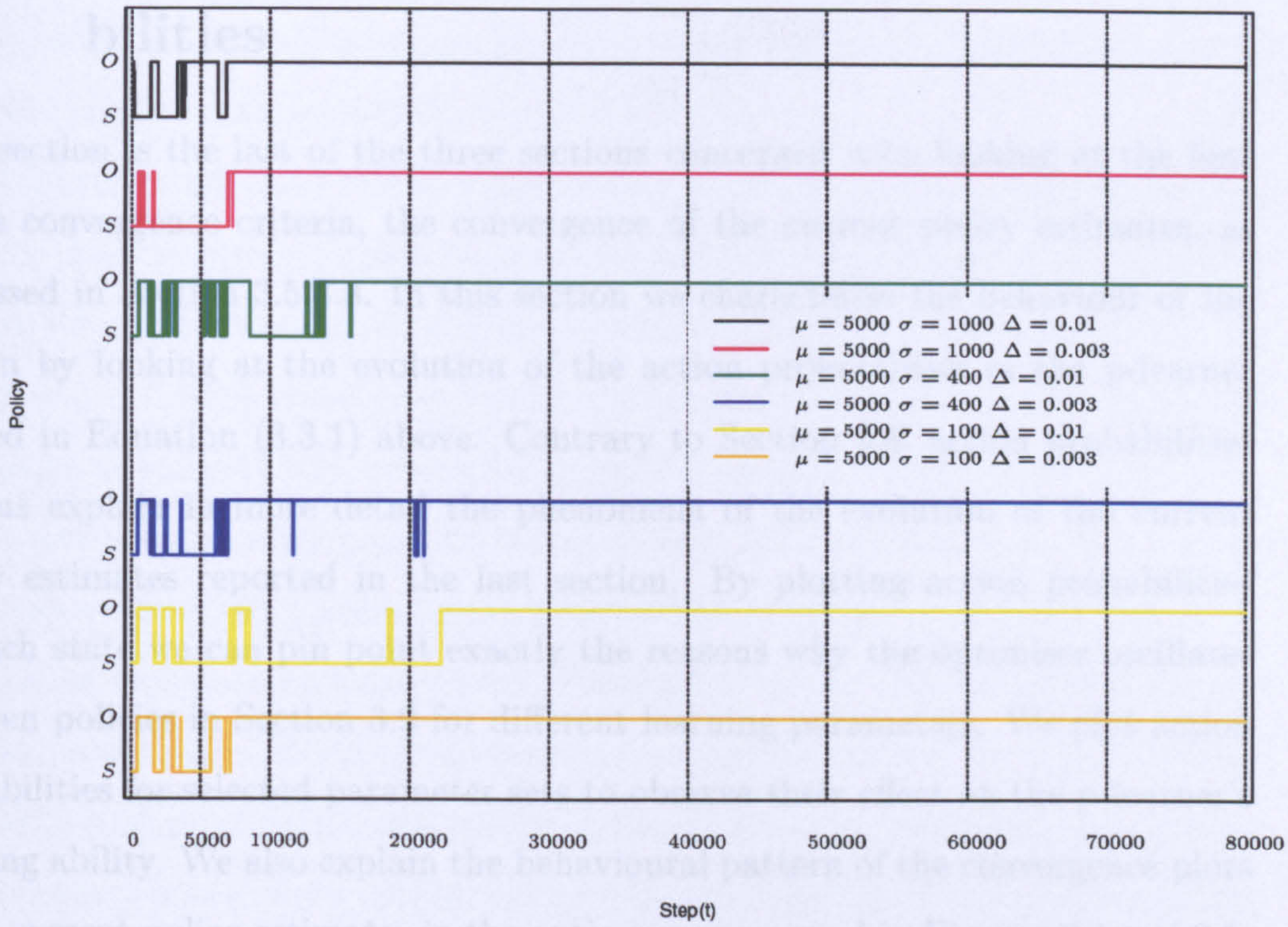


Figure 3.5: The graph above shows us the time intervals when the optimiser's (when concurrently run with the p-learner) current policy estimates converged to the true optimal policy (O) and a suboptimal policy (S) respectively for the different combinations of parameters $\mu = 5000$ $\sigma = 100, 400, 1000$ and $\Delta = 0.003, 0.01$ and $T = 80000$ for seed 1. We can see in all parameter sets the current policy estimate oscillated between the optimal policy and a suboptimal policy.

3.10 Running the optimiser concurrently with the p-learner - Plotting the Action Probabilities

This section is the last of the three sections concerned with looking at the first of the convergence criteria, the convergence of the current policy estimates, as discussed in Section 3.5.3.3. In this section we characterise the behaviour of the system by looking at the evolution of the action probabilities in the p-learner defined in Equation (3.3.1) above. Contrary to Section 3.8, action probabilities help us explain in more detail the phenomena of the evolution of the current policy estimates reported in the last section. By plotting action probabilities for each state we can pin point exactly the reasons why the optimiser oscillates between policies in Section 3.9 for different learning parameters. We plot action probabilities for selected parameter sets to observe their effect on the p-learner's learning ability. We also explain the behavioural pattern of the convergence plots of the current policy estimates in the optimiser, presented in Figures 3.4 and 3.5, using the action probability plots below.

The values of the learning parameters considered in this section using Method 3 are $\mu = 5000, 10000, 20000$, $\sigma = 100$ and 400, and $\Delta = 0.003$ and 0.01; again with an extra σ equal to 1000 added for $\mu = 5000$ (see Section 3.5.1). We recall that μ controls the amount of learning we are willing to do, σ controls how quickly we move from uniform sampling to focused sampling and Δ controls the rate at which we discriminate between action $u \in U(i)$ in state $i \in S$ after time μ. Note the bigger the value of σ the slower the transition between uniform sampling to focused sampling, and the bigger value of Δ the faster the rate of discrimination between actions after time μ. In this section only single runs are considered (see Section 3.5.3.1).

The figures labelled from 3.6 to 3.15 below are plots of the action probabilities

using different learning parameter sets. Note for each learning parameter set the action probabilities, defined in Equation (3.3.1), are only calculated and plotted at the current state $i \in S$ and included actions $u \in U(i)$ at time t. Thus the graphs presented for each state are plotted at irregularly spaced intervals because we only visit one state at time t.

To illustrate the kind of effect μ can have on the p-learner's learning ability to make decisions, we compare Figures 3.6 and 3.7 with Figures 3.10 and 3.11. Figures 3.6 and 3.7 look at the action probability plots for states $\{i = 0, 1, 2, 3, 4\}$ and $\{i = 5, 6, 7, 8, 9\}$ respectively using the learning parameter set $\{\mu = 20000$, $\sigma = 400.0$ and $\Delta = 0.01\}$, whereas Figures 3.6 and 3.7 on the other hand look at the action probability plots for states $\{i = 0, 1, 2, 3, 4\}$ and $\{i = 5, 6, 7, 8, 9\}$ respectively using the learning parameter set $\{\mu = 10000$, $\sigma = 400.0$ and $\Delta = 0.01\}$. If we look back to Figure 3.4 in Section 3.9 we see that the current policy estimate eventually settled down to the true optimal policy for $\{\mu = 20000$, $\sigma = 400.0$ and $\Delta = 0.01\}$, but not for $\{\mu = 10000$, $\sigma = 400.0$ and $\Delta = 0.01\}$. These results correspond to the action probability plots below and the proportion of time spent sampling individual actions over each state in Table 3.6.

In the "easy states", both learning parameter sets show that the p-learner had no difficulties focusing on the true optimal actions after the point of discrimination (time μ). However, in the "hard states", states 4 and 5, the system took a little longer to separate the current action estimates from the second best actions, but the p-learner eventually focused on the true optimal actions in these states with probability 1. It was only for $\{\mu = 10000$, $\sigma = 400.0$ and $\Delta = 0.01\}$ in state 8, "the very difficult state", that the p-learner chose action 2 instead of the optimal action, action 1. This is because the current Q-values evaluated at actions 1 and 2 were to close, and the p-learner had not allowed itself a sufficient amount of time for exploration using such a small μ. The Q-values for the best two actions in state 8 were also close for $\{\mu = 20000$, $\sigma = 400.0$ and $\Delta = 0.01\}$, but we can see that the p-learner had gained enough information by the point

of discrimination to recover. Even at $t = 20000$, for $\{\mu = 10000, \sigma = 400.0$ and $\Delta = 0.01\}$, the system oscillated between the two actions in state 8 but eventually the current action settled down to action 2. If we compare this behaviour with that of the evolution of the current policy estimate, for the same set of parameters, we can see that they are similar after time μ.

Note all the results obtained corresponding to $\mu = 20000$, for different learning parameter sets, were similar both in terms of the action probabilities and the current policy estimate plots in Figure 3.4. This is because for $\mu = 20000$ the p-learner sampled over all the actions with equal probability in each state for approximately 20000 iterations; and $\widetilde{\pi}_t(\cdot)$ equalled $\pi^*(\cdot)$ by $t = 8909$. This is partly true of all the learning sets corresponding to $\mu = 10000$. For $\{\mu = 20000$, $\sigma = 400.0$ and $\Delta = 0.01\}$, we can see that at first the p-learner chose the same actions as the $\mu = 20000$ case, therefore Figure 3.4 shows that the current action estimates were the same to begin with.

If we compare the evolution of the current policy estimates for the learning parameters $\{\mu = 10000, \sigma = 400.0$ and $\Delta = 0.01\}$ with $\{\mu = 10000, \sigma = 400.0$ and $\Delta = 0.003\}$, we see that the current policy estimates settled down to the true optimal policy for the latter set but not the former set. We plotted action probabilities for the two parameter sets to find the effect of decreasing Δ, from 0.01 to 0.003, had on the p-learner's ability to make the correct decision in each state. To do this we compare Figures 3.10 and 3.11 with 3.8 and 3.9 respectively, where Figures 3.8 and 3.9 represent the action probability plots for states $\{i = 0, 1, 2, 3, 4\}$ and $\{i = 5, 6, 7, 8, 9\}$ respectively using the parameter set $\{\mu = 10000$, $\sigma = 400.0$, $\Delta = 0.003\}$.

From Figures 3.10 and 3.11 we can see that the p-learner focused on the optimal actions a little sooner in the "easy states" compared with Figures 3.8 and 3.9, but in the "hard states" we see a significant difference - especially in state 4. In state 4 (for parameters $\{\mu = 10000, \sigma = 400.0$ and $\Delta = 0.003\}$) the p-learner did not choose the current action estimate with probability 1, as eventually

happened using $\{\mu = 10000, \sigma = 400.0 \text{ and } \Delta = 0.01\}$, but the probability tended to 1. In state 8, using the learning set $\{\mu = 10000, \sigma = 400.0 \text{ and } \Delta = 0.003\}$, the p-learner chose the optimal action asymptotically with probability 1 which was an improvement but it struggled on the way. This again happened because the Q-values in the two best actions were extremely close, but the decrement in Δ meant that the little learning we did after the point of discrimination was enough for the p-learner to recover. Lets compare the evolution of the action probabilities in state 8 with that of the current policy estimate. We can see by comparing these plots that in the time period when the p-learner was indecisive about which action to choose (after time μ) in state 8, the optimiser had trouble deciding which policy to choose. This is not surprising since the current action estimates depend on the estimates of the state transition probabilities. We compared the effect of using different values of Δ, in other learning parameter sets, and the same qualitative results were found (see Table 3.8). However it must be said when we compared different values of Δ for $\mu = 10000$ and $\sigma = 100$, the p-learner did do enough learning in both, since σ was very small (see Tables 3.8).

We then reduced the value of μ using different values of σ to illustrate the effect they had on the p-learner's decision to sample optimal actions. We chose to use parameters $\mu = 5000$, $\sigma = 400, 1000$, and $\Delta = 0.01$. Figures 3.12 and 3.13 below look at the action probability plots for states $\{i = 0, 1, 2, 3, 4\}$ and $\{i = 5, 6, 7, 8, 9\}$ respectively using the learning set $\{\mu = 5000, \sigma = 400.0, \Delta = 0.01\}$, whereas subsequent figures, Figures 3.14 and 3.15 look at the action probability plots for states $\{i = 0, 1, 2, 3, 4\}$ and $\{i = 5, 6, 7, 8, 9\}$ respectively using the learning parameter set $\mu = 5000$, $\sigma = 1000.0$, $\Delta = 0.01$. The function $\gamma'(t)$ stays closer to its asymptote for $\sigma = 400$ than for $\sigma = 1000$. Consequently, the four figures show that the p-learner sampled actions uniformly for longer in each state for $\sigma = 400$ compared to $\sigma = 1000$. Using $\sigma = 1000$, perhaps the method was forced to make a decision far too early, but such a large value of σ delayed the switch from uniform sampling to focused sampling. In both cases the

current action estimates focused on true optimal actions in each state $i \in S$. This corresponds to the results presented in Table 3.9 and Figure 3.5 in the previous two sections.

For both parameter sets the p-learner sampled the current action estimates in the "easy states" with ease. The p-learner also chose the optimal actions in the "hard states" with no major difficulties, but it found it harder to focus on them compared with the easy states. However, to begin with, the p-learner struggled with state 8 but it eventually chose the right action with probability 1. If we compare the evolution of the action probabilities for state 8, for both parameter sets, with that of the evolution of the current policy estimates after time μ we can see again the behavioural pattern is the same. We have seen from Table 3.10 that in general the p-learner struggled when μ was small, especially when σ was large. We recall that the current action estimate in state 8 converged to the true optimal action only 13 times out of the 20 simulations when we set $\mu = 5000$, $\sigma = 1000$ and $\Delta = 0.01$. Note the lower the value of μ the less the amount of time is allocated to total sampling. Hence in Figure 3.5 we see that when $\mu = 5000$ was used the optimiser oscillated between policies in different time intervals compared with when μ equalled 20000 and 10000 (see Figure 3.4).

Concluding, in the "easy states" we saw that we had learnt enough information about the system to focus on the true optimal actions in every parameter set. Thus a value of σ close to zero or even a large value of σ would have worked equally as well. The p-learner does all the work it needs to do before the discrimination stage.

We have seen Δ is most effective in the hard states. The parameter is important when we have not yet done enough learning by $t = \mu$ - in which case it may be helpful to keep Δ small. We saw this when we compared the learning parameter set $\{\mu = 10000, \sigma = 400, \Delta = 0.01\}$ with $\{\mu = 10000, \sigma = 400, \Delta = 0.003\}$.

From the above simulations we have also seen that if μ is small we are in danger of taking a wrong decision far too early before we have accumulated

enough experience to discriminate. We originally thought it may be better to take the decision more slowly by taking a larger σ, but we have seen taking a large value of σ still has serious consequences as shown in Table 3.10. This is not necessarily because the value of σ is too big, it is again because μ is too small. In order to find the true optimal policy we need to sample over all the actions in each state to gain good estimates of both the Q-values, evaluated at each state and action, and also the true state transition probabilities. Obviously, $\mu = 5000$ is too small a value in which to do this.

In the remaining chapter we will try and answer the question, does it really matter if we focus on a suboptimal action over the optimal one in a state where the Q-values for these two actions are close?

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

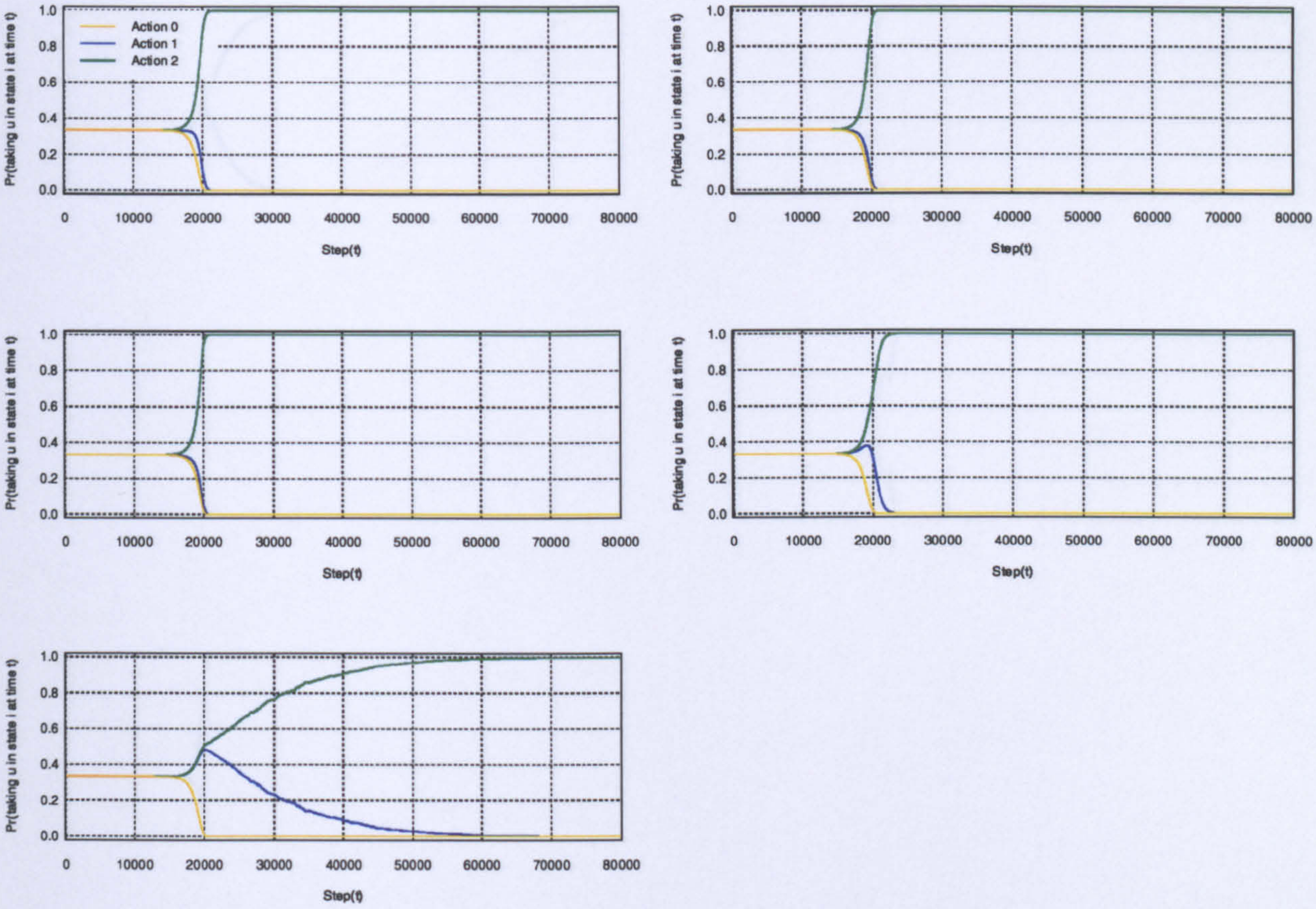


Figure 3.6: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states 0, 1, 2, 3 and 4, are presented from left to right, top to bottom using parameters $\mu = 20000$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

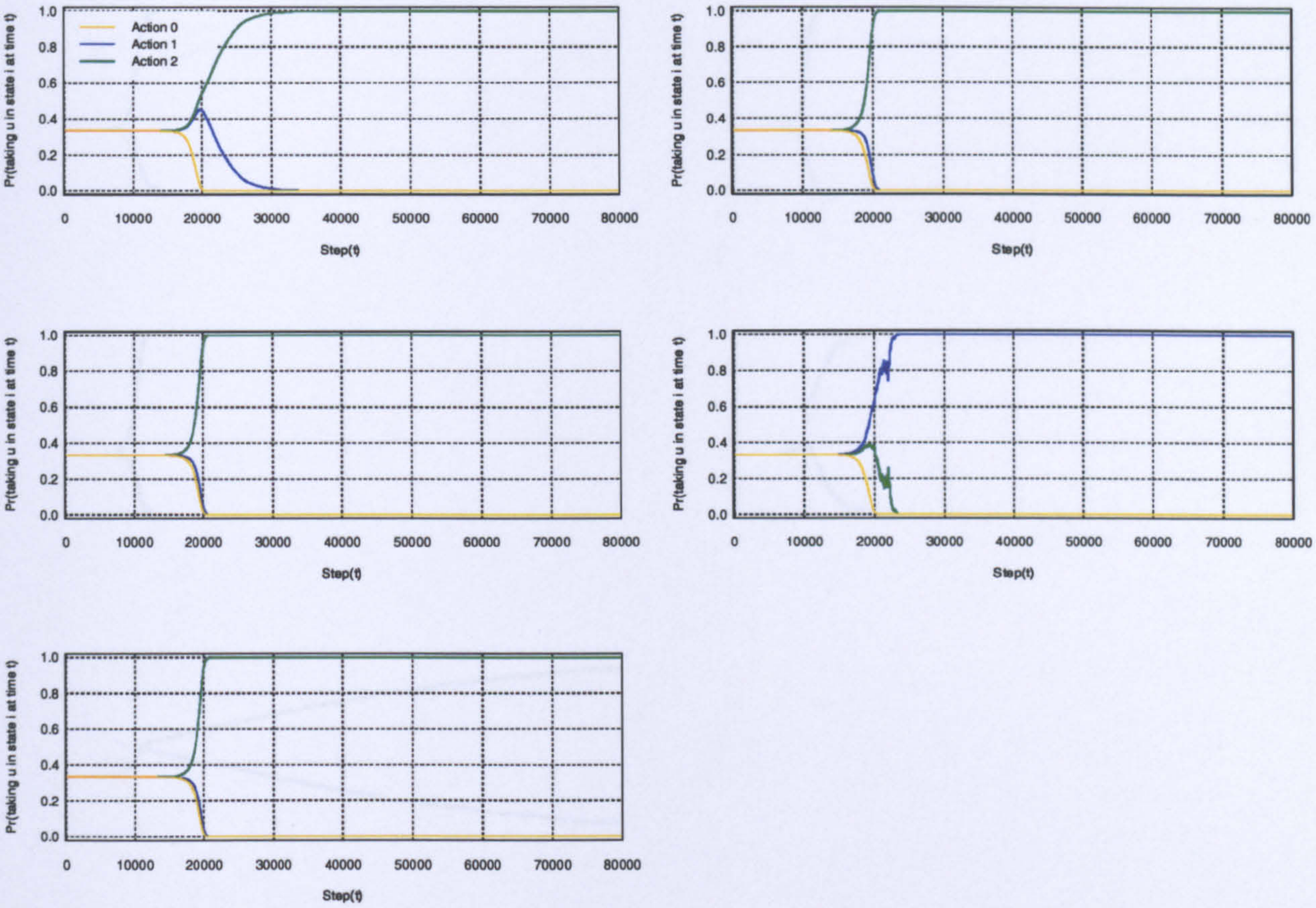


Figure 3.7: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states $5, 6, 7, 8$ and 9, are presented from left to right, top to bottom using parameters $\mu = 20000$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

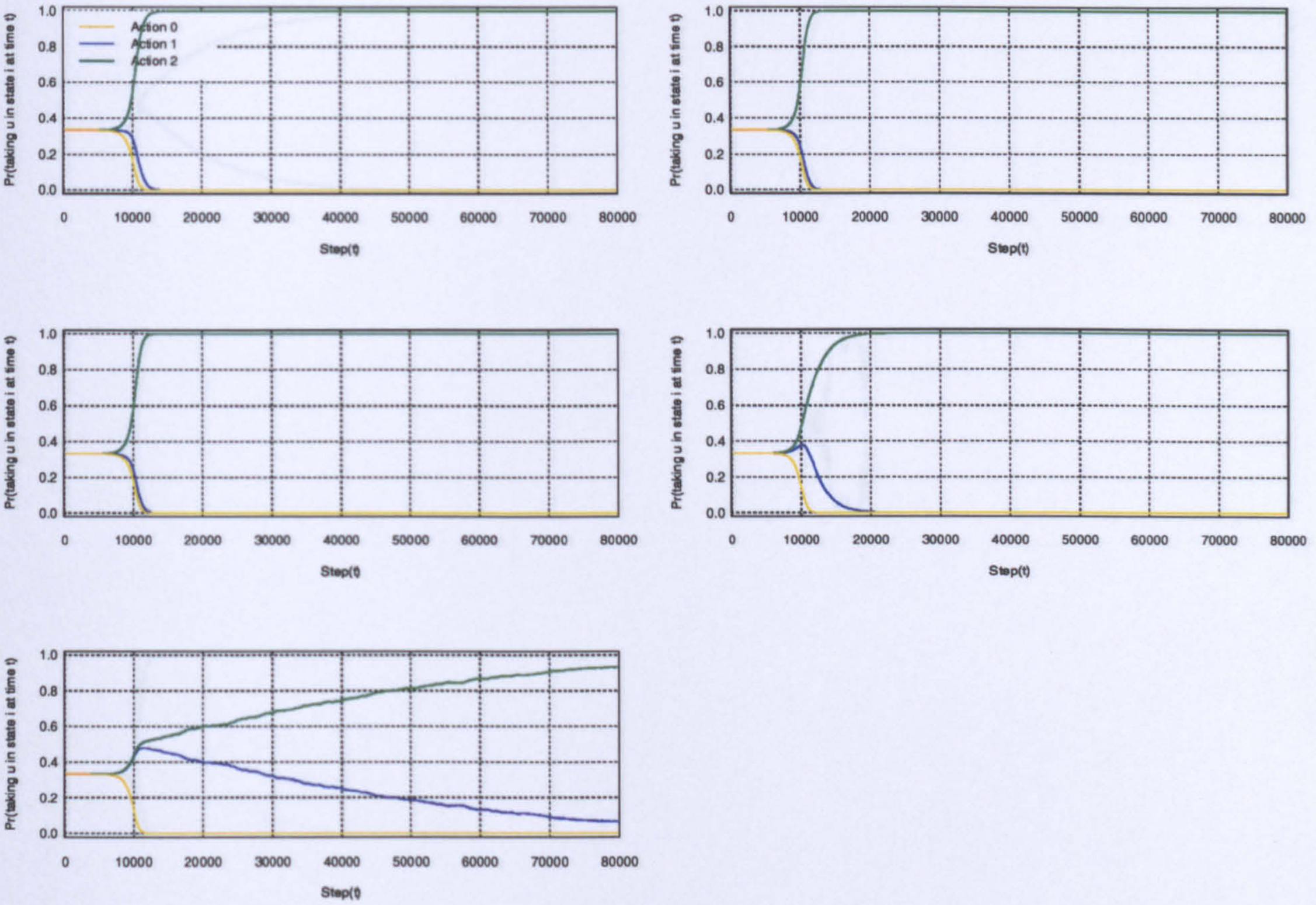


Figure 3.8: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states $0, 1, 2, 3$ and 4, are presented from left to right, top to bottom using parameters $\mu = 10000$, $\sigma = 400.0$, $\Delta = 0.003$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

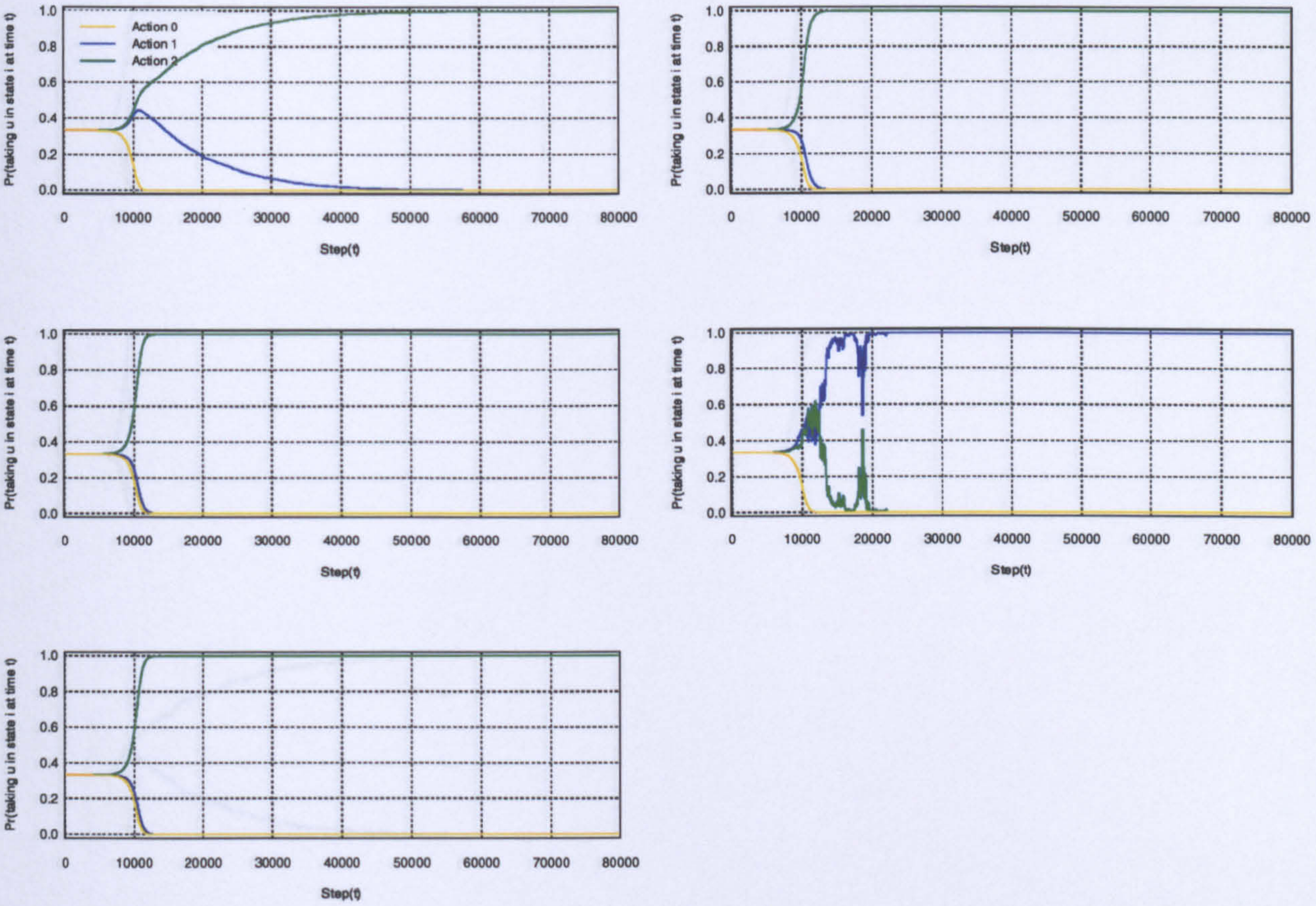


Figure 3.9: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states $5, 6, 7, 8$ and 9, are presented from left to right, top to bottom using parameters $\mu = 10000$, $\sigma = 400.0$, $\Delta = 0.003$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

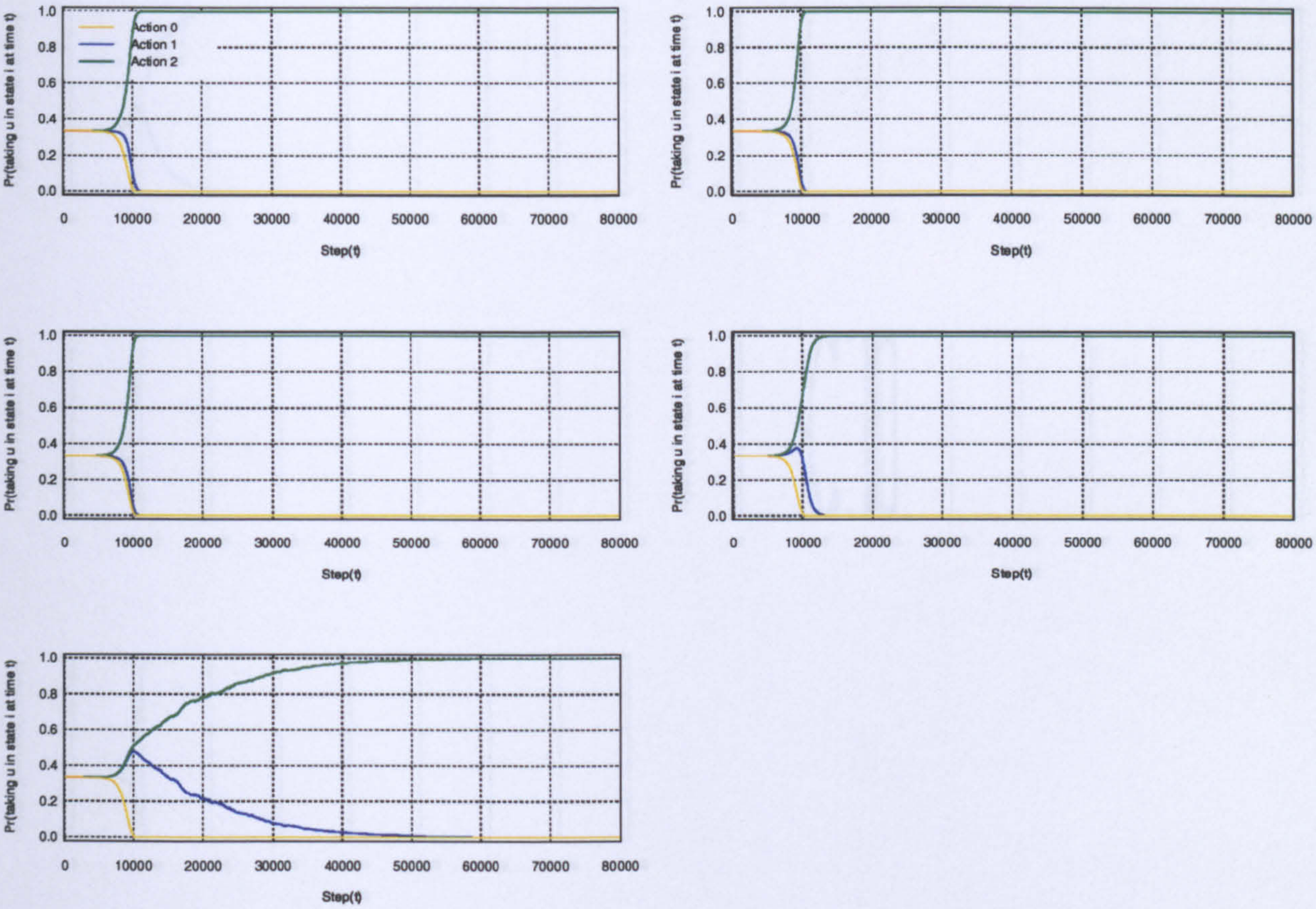


Figure 3.10: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states 0, 1, 2, 3 and 4, are presented from left to right, top to bottom using parameters $\mu = 10000$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

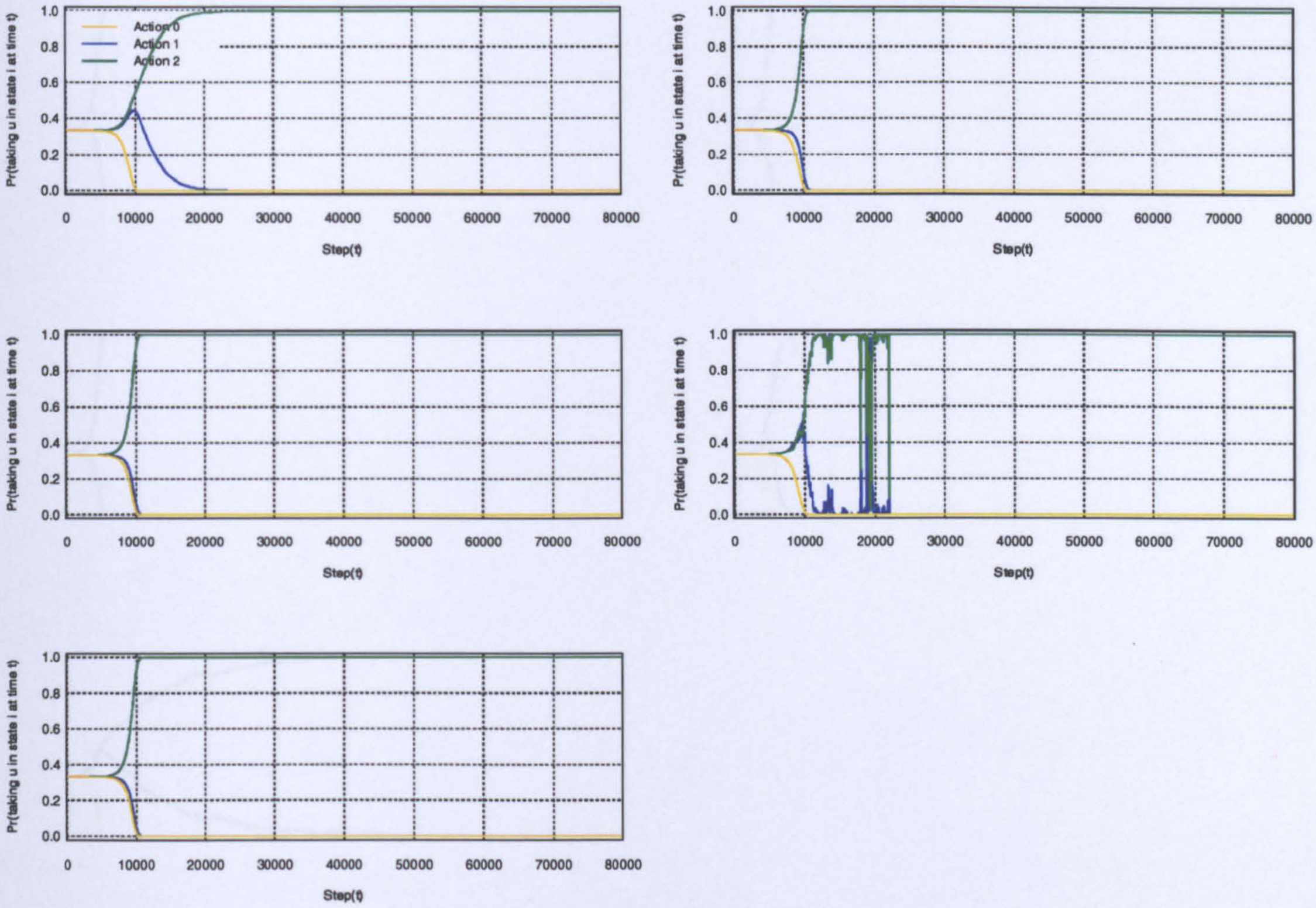


Figure 3.11: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states $5, 6, 7, 8$ and 9, are presented from left to right, top to bottom using parameters $\mu = 10000$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

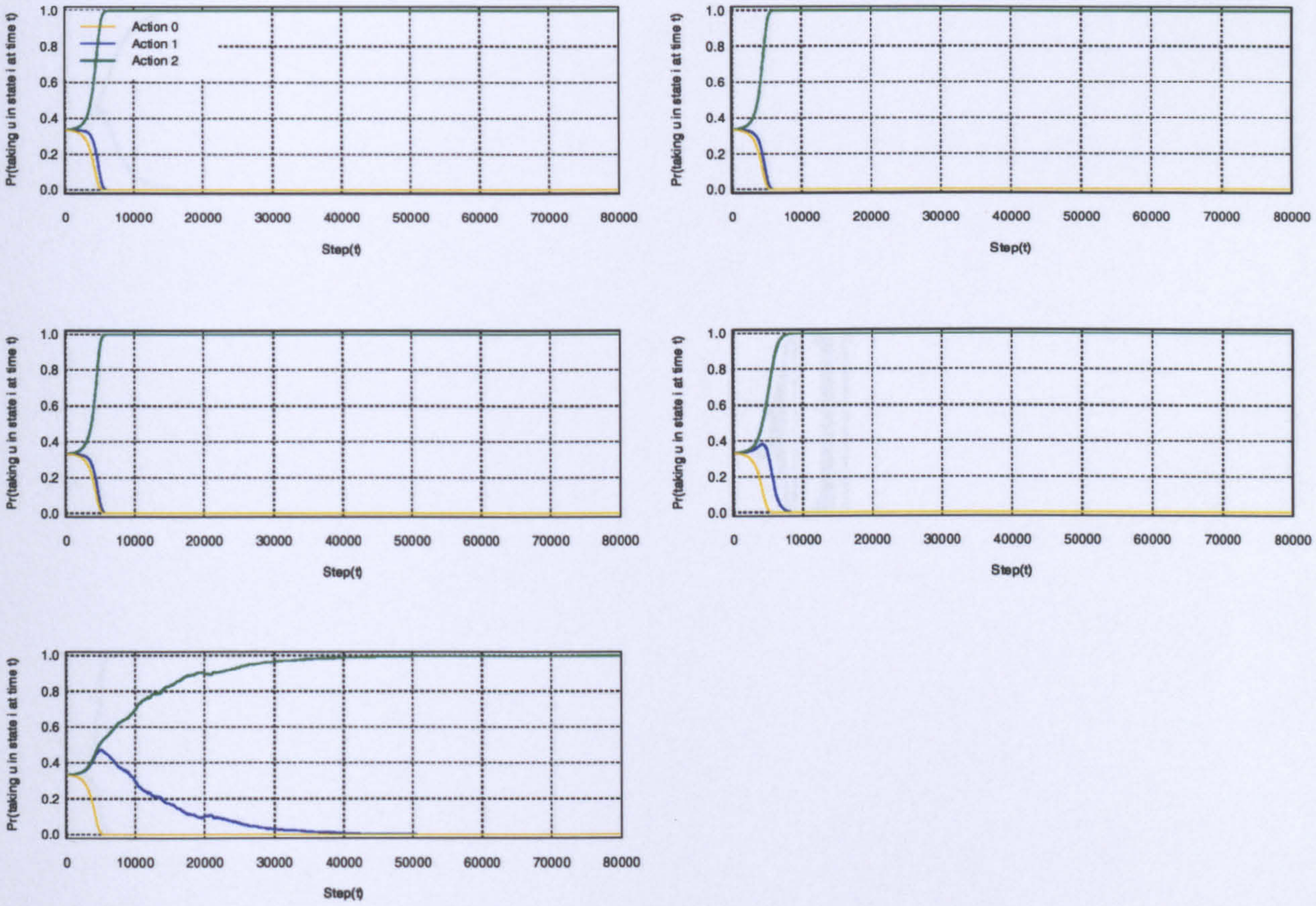


Figure 3.12: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states $0, 1, 2, 3$ and 4, are presented from left to right, top to bottom using parameters $\mu = 5000$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

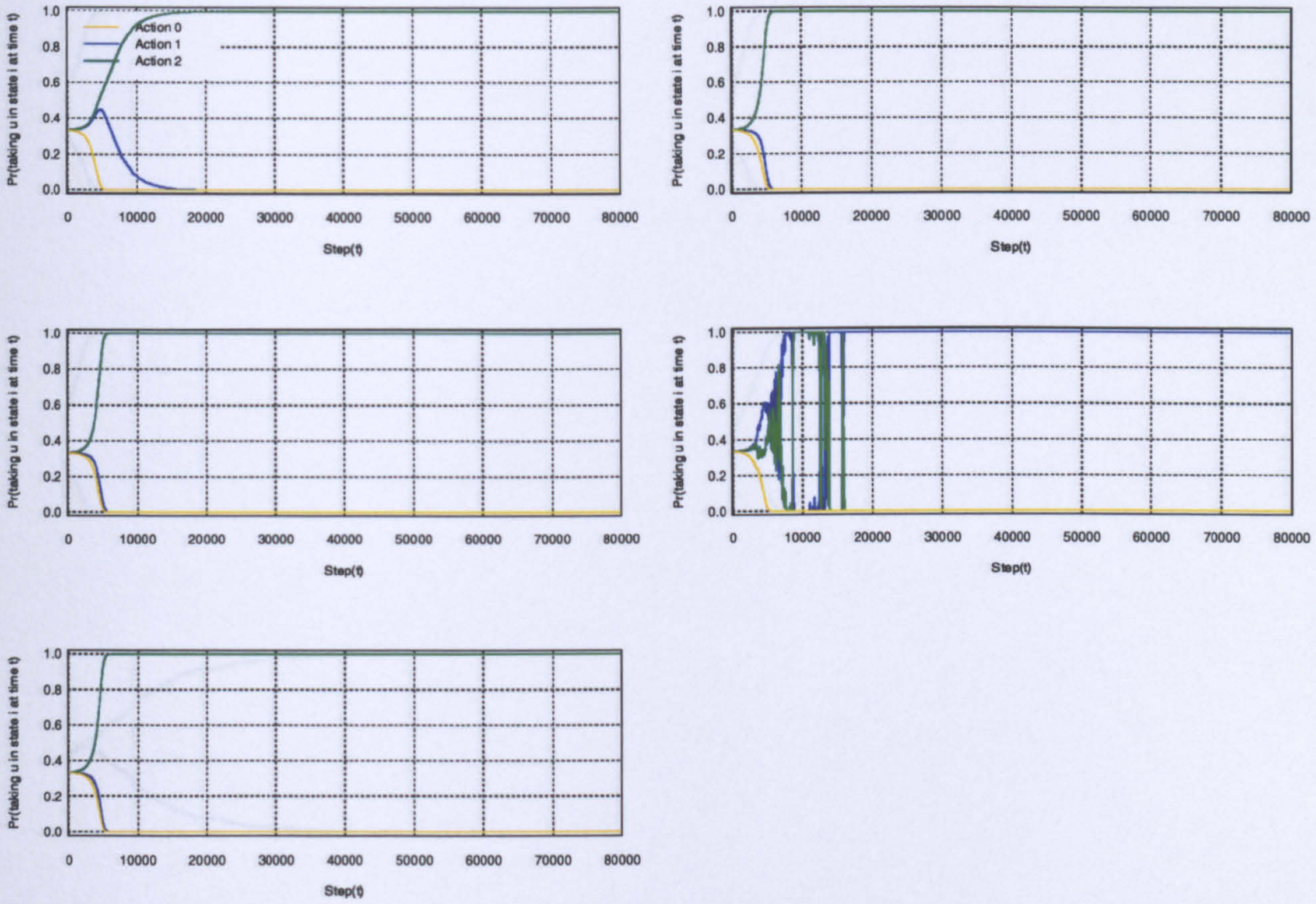


Figure 3.13: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states $5, 6, 7, 8$ and 9, are presented from left to right, top to bottom using parameters $\mu = 5000$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the P-learner - Plotting the Action Probabilities

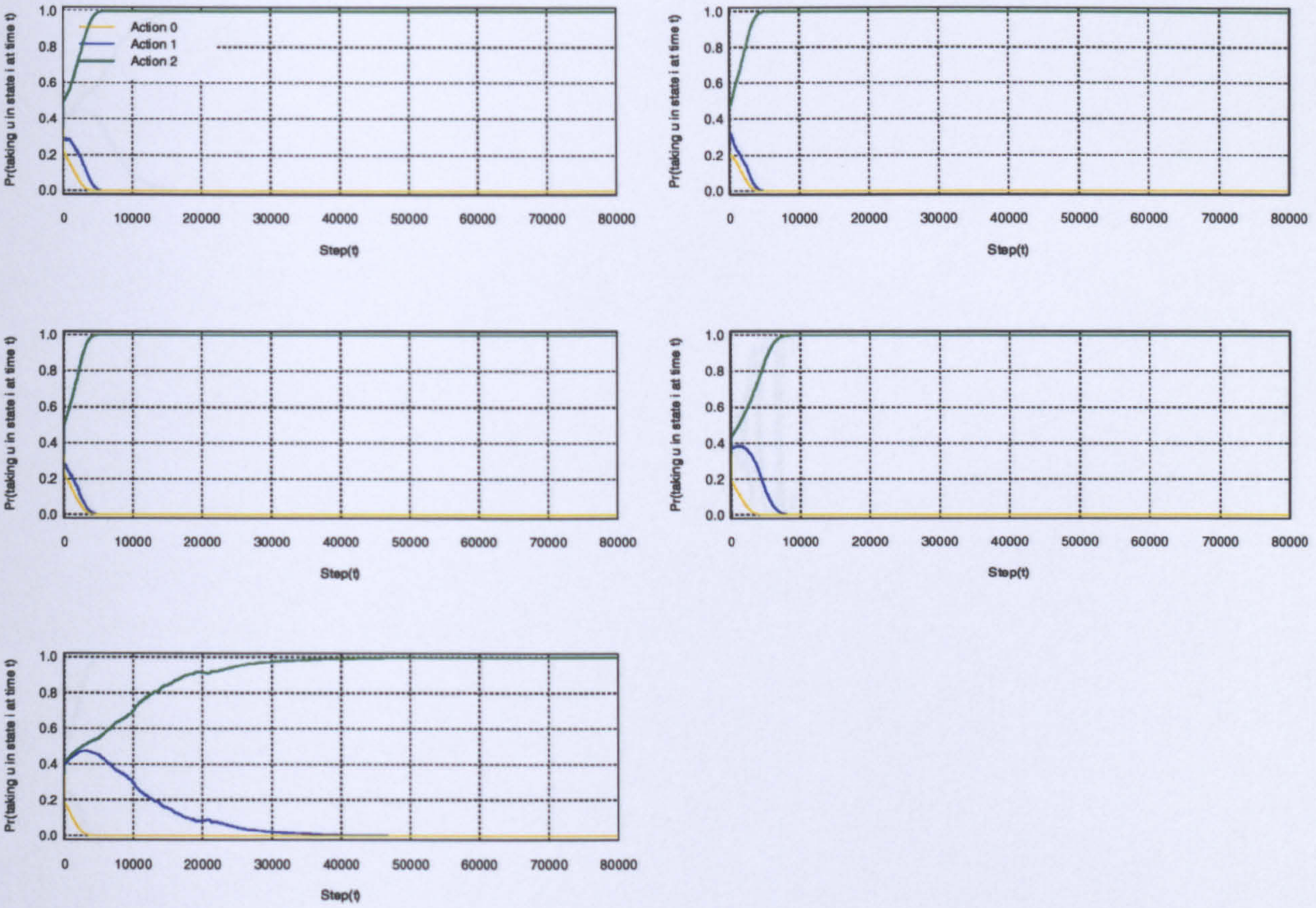


Figure 3.14: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states $0, 1, 2, 3$ and 4, are presented from left to right, top to bottom using parameters $\mu = 5000, \sigma = 1000.0, \Delta = 0.01$ and $T = 80000$ with seed 1.

3.10 Running the optimiser concurrently with the p-learner - Plotting the Action Probabilities

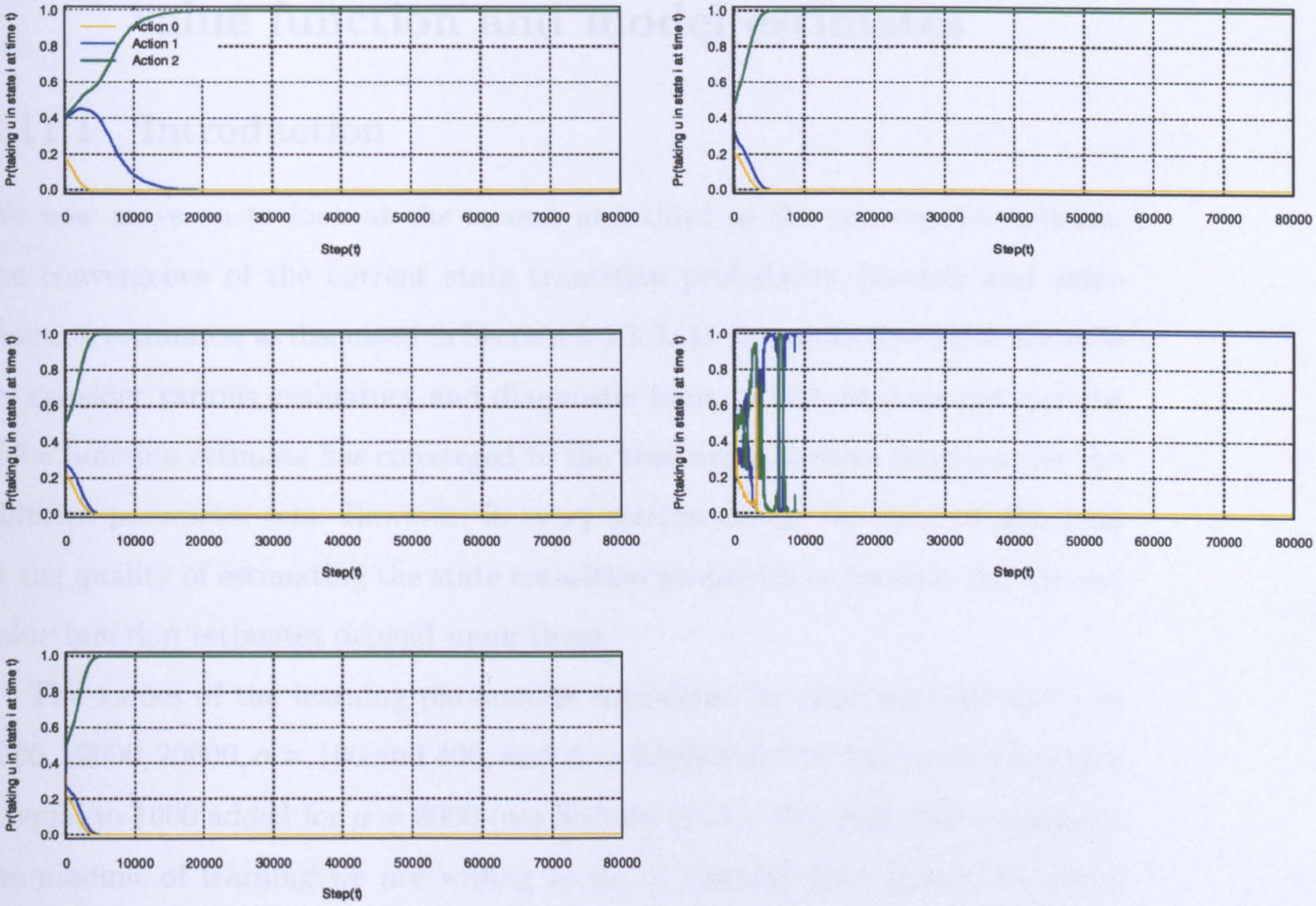


Figure 3.15: The graph above is a plot of the probability frequencies (action probabilities) of taking an action $u \in U(i)$ in a particular state i plotted at irregularly spaced points (Step time (t)). The states $5, 6, 7, 8$ and 9, are presented from left to right, top to bottom using parameters $\mu = 5000$, $\sigma = 1000.0$, $\Delta = 0.01$ and $T = 80000$ with seed 1.

3.11 Running the optimiser concurrently with the p-learner - The evolution of the current value function and model estimates

3.11.1 Introduction

We now move on to look at the second and third of the convergence criteria, the convergence of the current state transition probability (model) and value function estimates, as discussed in Section 3.5.3.3. In this and subsequent sections we consider various evaluatory and diagnostic tools to test whether our current value function estimate has converged to the true optimal value function, for the different parameter sets. However, in every section except the last, we also look at the quality of estimating the state transition probabilities because the current value function estimates depend upon them.

The values of the learning parameters considered in these analyses are $\mu = 5000, 10000, 20000$, $\sigma = 100$ and 400, and $\Delta = 0.003$ and 0.01; again with an extra σ equal to 1000 added for $\mu = 5000$ (see Section 3.5.1). We recall that μ controls the amount of learning we are willing to do, σ controls how quickly we move from uniform sampling to focused sampling and Δ controls the rate at which we discriminate between action $u \in U(i)$ in state $i \in S$ after time μ. Note the larger the value of σ the slower the transition from uniform sampling to focused sampling, and the bigger value of Δ the faster the rate of discrimination between actions after time μ. In this and subsequent sections both single and multiple runs are considered depending upon the required output (see Section 3.5.3.1).

To compare the two criteria we plotted the root mean square error of the current value function estimates for all $i \in S$, denoted by $E_{\tilde{v}}(t)$, against the root mean square error of all the current state-transition probability estimates for all $i, j \in S$ and $u \in U(i)$, denoted by $E_{\hat{p}}(t)$, parameterised at both time t and at log

to the base 10 where,

$$E_{\tilde{v}}(t) = \sqrt{\left\{\sum_{i \in S} \{v(i) - \tilde{v}_t(i)\}^2 \{|S|\}^{-1}\right\}}, \tag{3.11.5}$$

and

$$E_{\hat{p}}(t) = \sqrt{\left\{\sum_{i \in S} \sum_{j \in S} \sum_{u \in U(i)} \{p_{ij}(u) - \hat{p}^t_{ij}(u)\}^2 \{|S|^2|U|\}^{-1}\right\}}. \tag{3.11.6}$$

To begin with the current value function estimates in all of the states were set to zero and the current state transition probability estimates had a uniform prior, therefore both sets of errors were very large at time 0. Thus the figures below comparing $\log_{10}(E_{\tilde{v}}(t))$ with $\log_{10}(E_{\hat{p}}(t))$ start at the top right hand side of the graphs before eventually making their way over to the left hand side, where it is hoped the p-learner has learnt enough information about the true state transition probabilities and the optimiser has indirectly learnt about the true optimal value function, for $E_{\tilde{v}}(t)$ to tend to zero.

For the current value function estimates to converge to the true optimal value functions in each state $i \in S$, we have to focus on obtaining good estimates of the true state transition probabilities following an optimal policy. This takes time. For this reason if T is small the optimal value of μ will be a relatively big fraction of T. Therefore in the following figures comparing $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$, we plot tick marks at maximium intervals of 10000 iterations labelled from 1 to 8 to see the effects of varying T from 10000 to 80000, for a given value of μ.

Figures 3.16 and 3.17 below illustrate the evolution of the method's performance for learning parameter sets $\{\mu = 20000, \sigma = 400\ \Delta = 0.01\}$ and $\{\mu = 10000, \sigma = 400\ \Delta = 0.01\}$ respectively. One of the reasons behind choosing these parameter sets was that they give a good illustration of how much the p-learner learns about the overall model for different values of μ. The other is that (as previously shown in Figure 3.4) for $\{\mu = 20000, \sigma = 400\ \Delta = 0.01\}$ the

current policy estimate settled down to the optimal policy, but for $\{\mu = 10000, \sigma = 400, \Delta = 0.01\}$ it did not. Therefore by comparing the two, we will see whether choosing a suboptimal policy over an optimal one has a significant effect on the convergence of $\widetilde{v}_t(\cdot)$ to $v^*(\cdot)$. Each plot consists of two independent runs to see if the history of the process differs from one seed to the next. On the other hand, Figures 3.18 and 3.19 (one for each seed) summarise the performance of all of the learning parameter sets corresponding to $\mu = 20000$ and 10000. They show how sensitive the method's performance is for different values of σ and Δ. The above is repeated for $\mu = 5000$.

Figures 3.20 and 3.21 illustrate the evolution of the method's performance for learning parameter sets $\{\mu = 5000, \sigma = 400\ \Delta = 0.01\}$ and $\{\mu = 5000, \sigma = 1000\ \Delta = 0.01\}$ respectively. They are both used to compare how much the p-learner learns about the overall model and to see to what degree $\widetilde{v}_t(\cdot)$ converges to $v^*(\cdot)$. These figures are used for comparison purposes, amongst themselves , and with Figures 3.16 and 3.17. Figures 3.22 and 3.23 (one for each seed), on the other hand, report how sensitive the method's performance is for different values of σ and Δ, using such a small value of μ as 5000.

In all of the figures illustrating the evolution of the method's performance we note a consistent trend in the method's behaviour. Figure 3.24 below and Table 3.11 above, help us to explain this phenomena. Figure 3.24 illustrates the history of the process of the plot $\log_{10}(E_{\widetilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ for the learning set $\{\mu = 20000, \sigma = 400$ and $\Delta = 0.01\}$, but this time we labelled the method's performance at irregular time points from 100 to 80000 iterations, labelled from 0.01 to 8. Table 3.11, on the other hand, shows us the time intervals when the optimiser running concurrently with the p-learner chose the optimal action in each particular state.

We end Section 3.11 by comparing the performance of the optimiser when run concurrently with the p-learner with that of two other methods, namely, the optimiser on its own and the Pre-Jacobi method. We measure their performance

using Figure 3.25, plotting the error in the current value function estimates for each method against time step t.

3.11.2 Comparing errors in the current value function estimates with the current model estimates using $\mu = 20000$ and $\mu = 10000$

If we compare Figures 3.16 and 3.17 we can see that the p-learner learnt more about the overall model for $\mu = 20000$ compared with $\mu = 10000$. This makes sense because in the $\mu = 10000$ case the p-learner "homed-in" on the set state-transitions probabilities following the current policy estimate sooner than the $\mu = 20000$ case. Note that after the point of discrimination between actions (time μ), the errors in the current value function estimate $E_{\tilde{v}}(t)$ converged to zero quicker than when the p-learner sampled over all the actions in each state - again because the p-learner focused in on estimating the most important part of the model. Figures 3.16 and 3.17 also illustrate if the simulation horizon was set to a small value of T e.g. 10000, the optimiser's estimates of the optimal value function would have been far from the final solutions in both seeds. This is because the p-learner would only have had a short amount of time in which to; a) decide which actions were optimal in each state and, b) sample the current action estimates in each state a sufficient number of times (in the hope that they are the true optimal actions) to gain good estimates of the optimal value function.

In the $\mu = 20000$ case Figure 3.16 shows that the error in the current value function estimate was slightly larger at $t = 40000$ compared with that of $t = 30000$ using seed 1. This is because the p-learner had not sampled the state-transition probabilities following the current policy estimate enough times by $t = 30000$. If we look back to Figures 3.6 and 3.7, they show the p-learner did not sample the true optimal actions in all of states with probability 1 until some time after μ. Consequently, the p-learner did not have good enough estimates of

the important state-transition probabilities at $t = 30000$. Even after time T, the error defined in Equation (3.11.5) did not equal zero. For $E_{\tilde{v}}(t)$ to tend to zero $\log_{10}\{E_{\tilde{v}}(t)\}$ would have to tend to -6, and at time T it barely has not quite reached -1. This makes us wonder whether we are getting to the limit when we try to estimate the important state-transition probabilities, because Table 3.7 told us that we did visit each state often enough. What is encouraging, is that, the longer we sample the state transition probabilities following the current policy estimate the better our estimates of the true optimal value functions in each state $i \in S$. In both seeds we see a vast improvement in $\log_{10}\{E_{\tilde{v}}(T)\}$ compared with $\log_{10}\{E_{\tilde{v}}(70000)\}$. This was true of all learning parameter sets.

As for the $\mu = 10000$ case if we look back to Figure 3.11 we note that the optimiser's current action estimate in state 8 converged to action 2 instead of action 1, the true optimal action. Evidently the estimates of state-transition probabilities in the row corresponding to state 8 are far better for action 2 compared with that of action 1 - simply because the p-learner sampled all the elements in the row corresponding to state 8 in action 2 more frequently than for action 1. Therefore the optimiser's current value function estimate in state 8 consisted of the wrong current state-transition probability estimates, and as a consequence Figure 3.17 shows that $\log_{10}\{E_{\tilde{v}}(T)\}$ is bigger for seed 1 than in Figure 3.16.

We then plotted a summary of the performances running the optimiser concurrently with the p-learner for all the different sets of parameters corresponding to $\mu = 20000$ and 10000. The results of which are illustrated in Figures 3.18 and 3.19 below. As we can see the figures show nice, clear patterns, at least as far as the effects of μ and the seeds go, but they show a relative insensitivity to σ and Δ. This is due to the fact that the function $\gamma'(t)$ which we used in every state to discriminate between actions was a continuous function. The function began by choosing actions in each state with equal probability and then made an abrupt step at time μ with the intention of choosing the current action estimate in each state with probability 1. This is the sole reason why the history of the process is

the same for the most part, before time μ.

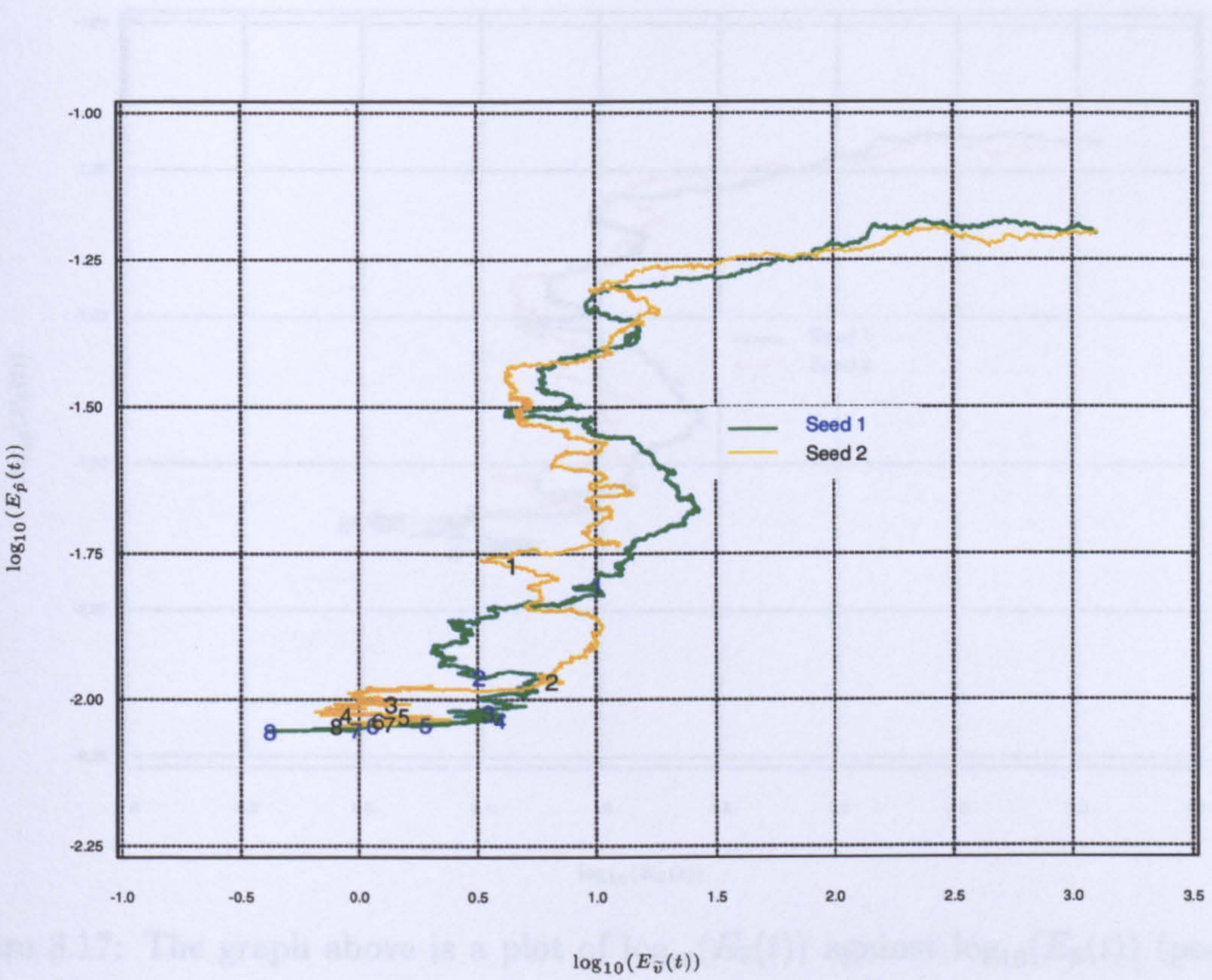


Figure 3.16: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; where $\mu = 20000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$. Twenty different seeds were used in all. We present results taken from two seeds to give us a good illustration of the method's performance. Tick marks were plotted for each of the two seeds at every 10000 iterations, labelled from 1 to 8.

3.11 Running the optimiser concurrently with the P-learner - The evolution of the current value function and model estimates

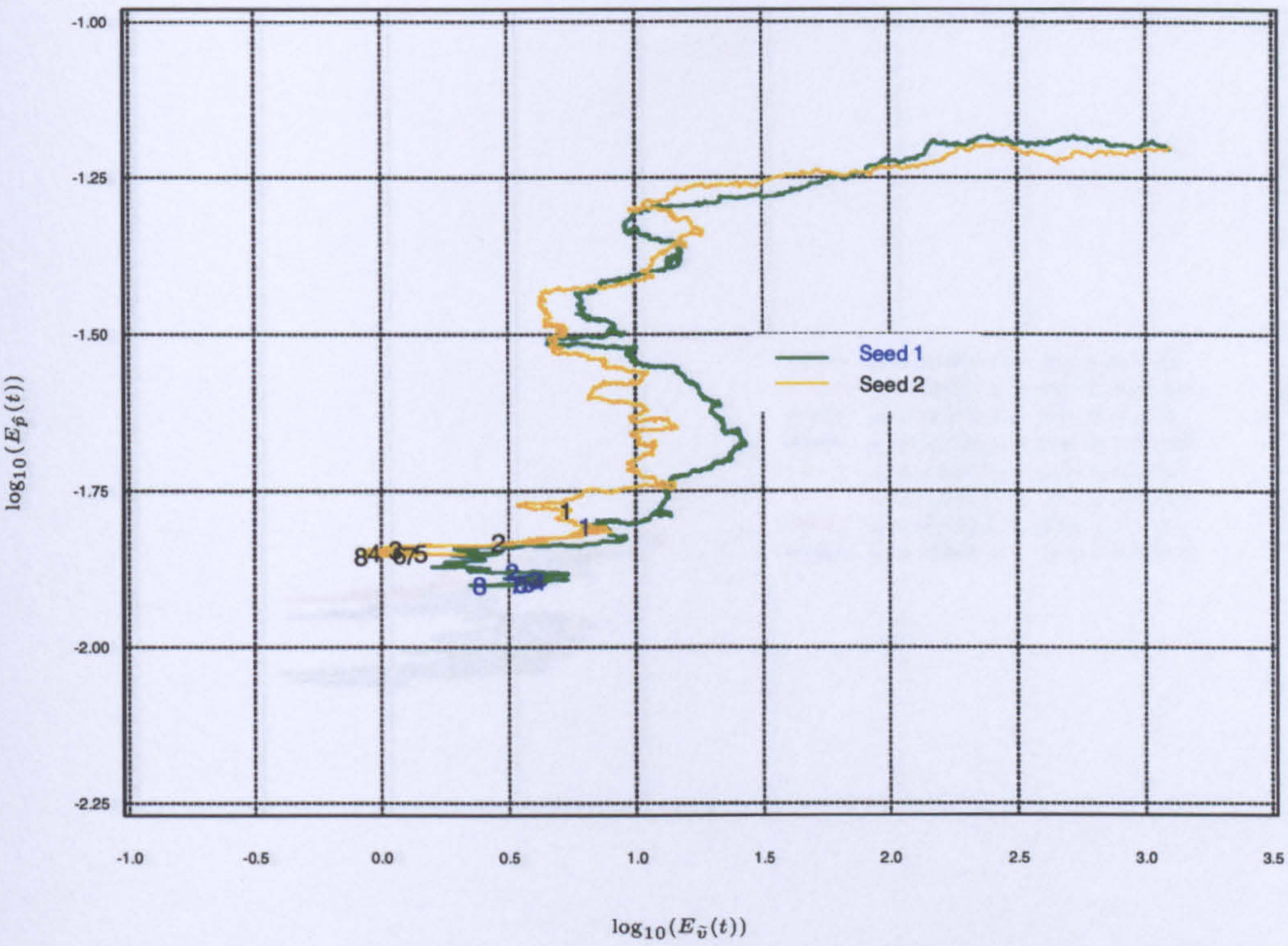


Figure 3.17: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; where $\mu = 10000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$. Twenty different seeds were used in all. We present results taken from two seeds to give us a good illustration of the method's performance. Tick marks were plotted for each of the two seeds at every 10000 iterations, labelled from 1 to 8.

3.11 Running the optimiser concurrently with the P-learner - The evolution of the current value function and model estimates

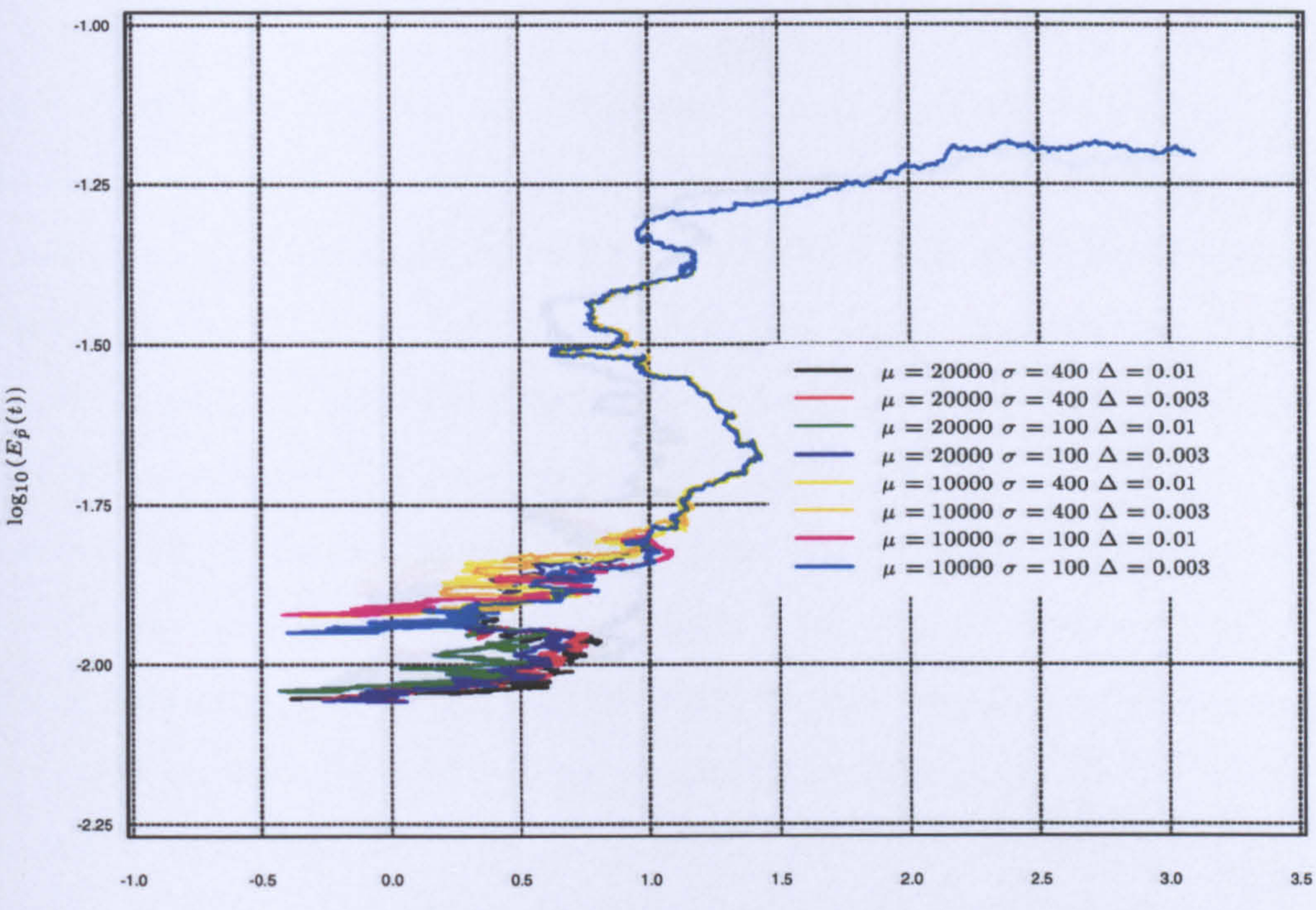


Figure 3.18: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for different values of μ, σ, and Δ with $T = 80000$. The graph illustrates results taken from seed 1.

3.11 Running the optimiser concurrently with the P-learner - The evolution of the current value function and model estimates

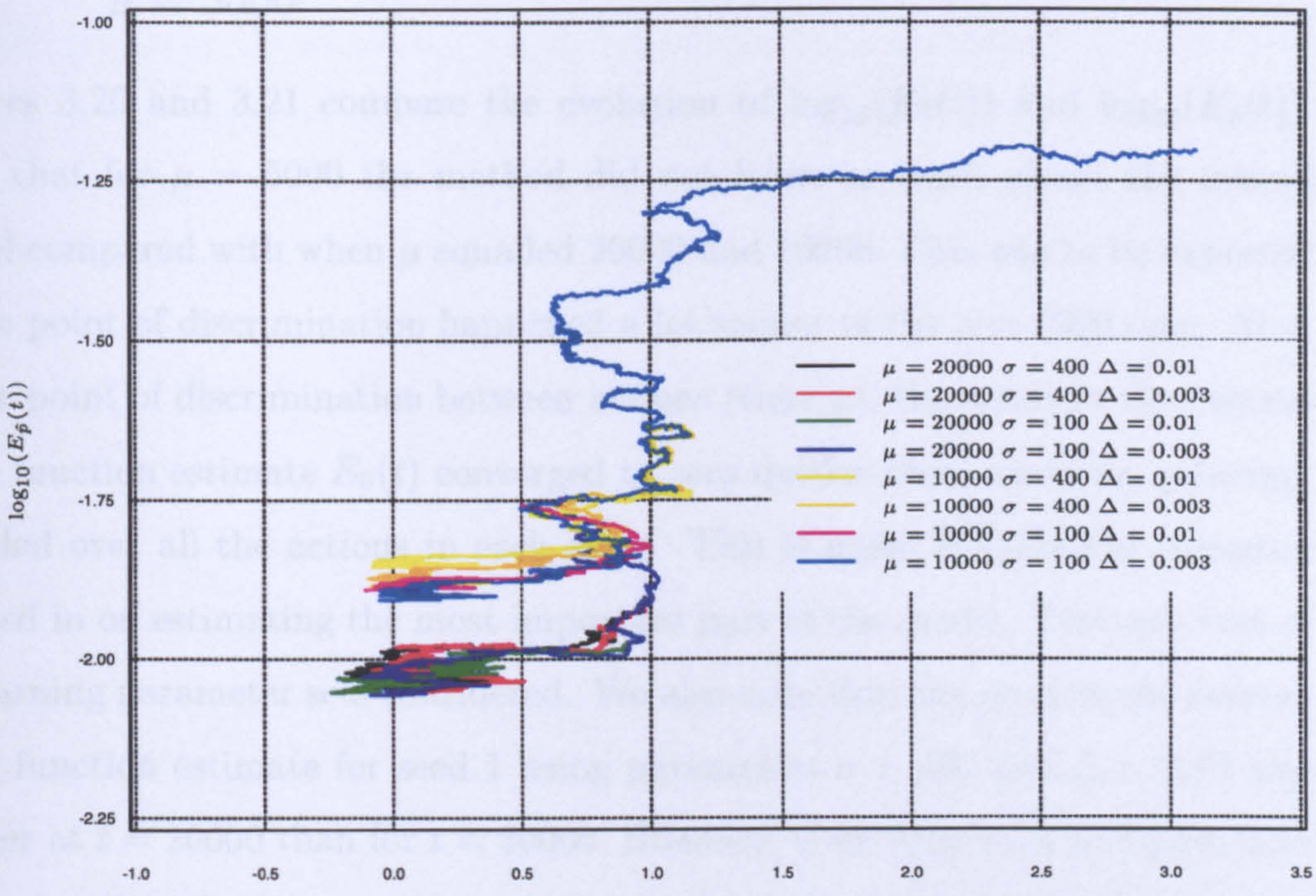


Figure 3.19: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for different values of μ, σ, and Δ with $T = 80000$. The graph illustrates results taken from seed 2.

3.11.3 Comparing the errors in the current value function estimates with the current model estimates using $\mu = 5000$

Figures 3.20 and 3.21 compare the evolution of $\log_{10}(E_{\tilde{v}}(t))$ and $\log_{10}(E_{\tilde{p}}(t))$. Note that for $\mu = 5000$ the method did not learn as much about the overall model compared with when μ equalled 20000 and 10000. This was to be expected as the point of discrimination happened a lot sooner in the $\mu = 5000$ case. Also, at the point of discrimination between actions (time μ), the errors in the current value function estimate $E_{\tilde{v}}(t)$ converged to zero quicker than when the p-learner sampled over all the actions in each state. This is again because the p-learner focused in on estimating the most important part of the model. This was true of all learning parameter sets considered. We also note that the error in the current value function estimate for seed 1 using parameters $\sigma = 400$ and $\Delta = 0.01$ was smaller at $t = 20000$ than for $t = 30000$. However, if we look back to Figure 3.13 we can see that the p-learner did not decide to sample the optimal action in state 8, or the hard states for that matter, until quite a late stage. A similar argument is true for the $\sigma = 1000$ case.

We note that the error in the current value function estimate, out of all the learning parameter sets considered in this experiment, was the smallest at time T. This is simply because the p-learner focused on the true optimal actions in each state earlier than was the case for the other parameter sets (see Figures 3.4 and 3.5). The figures also show $\log_{10}\{E_{\tilde{v}}(T)\}$ was far from -6 but, encouraging, that $\log_{10}\{E_{\tilde{v}}(T)\}$ was a big improvement on $\log_{10}\{E_{\tilde{v}}(70000)\}$.

We then plotted a summary of the performances for $\mu = 5000$ illustrated in Figures 3.22 and 3.23 for all the different sets of parameters, to compare the effect the different parameters had on the method's performance. The graphs show nice clear patterns at least as far as the effects of μ and seeds are concerned, but contrary to Figures 3.18 and 3.19 ($\mu = 20000$ and $\mu = 10000$) the graphs do

not show the relative insensitivity to σ and Δ. This is the case for two reasons: 1) the point of discrimination between actions happened quite early on and 2), using $\sigma = 1000$ meant $\gamma'(t)$ was further away from its asymptote compared with $\sigma = 400$ and 100. Consequently, the p-learner was only able to choose the same actions as $\sigma = 100$ and 400 at the start of the run. Hence the history of the method's performance for different learning parameters, shown in Figures 3.22 and 3.23, was unique.

3.11 Running the optimiser concurrently with the P-learner - The evolution of the current value function and model estimates

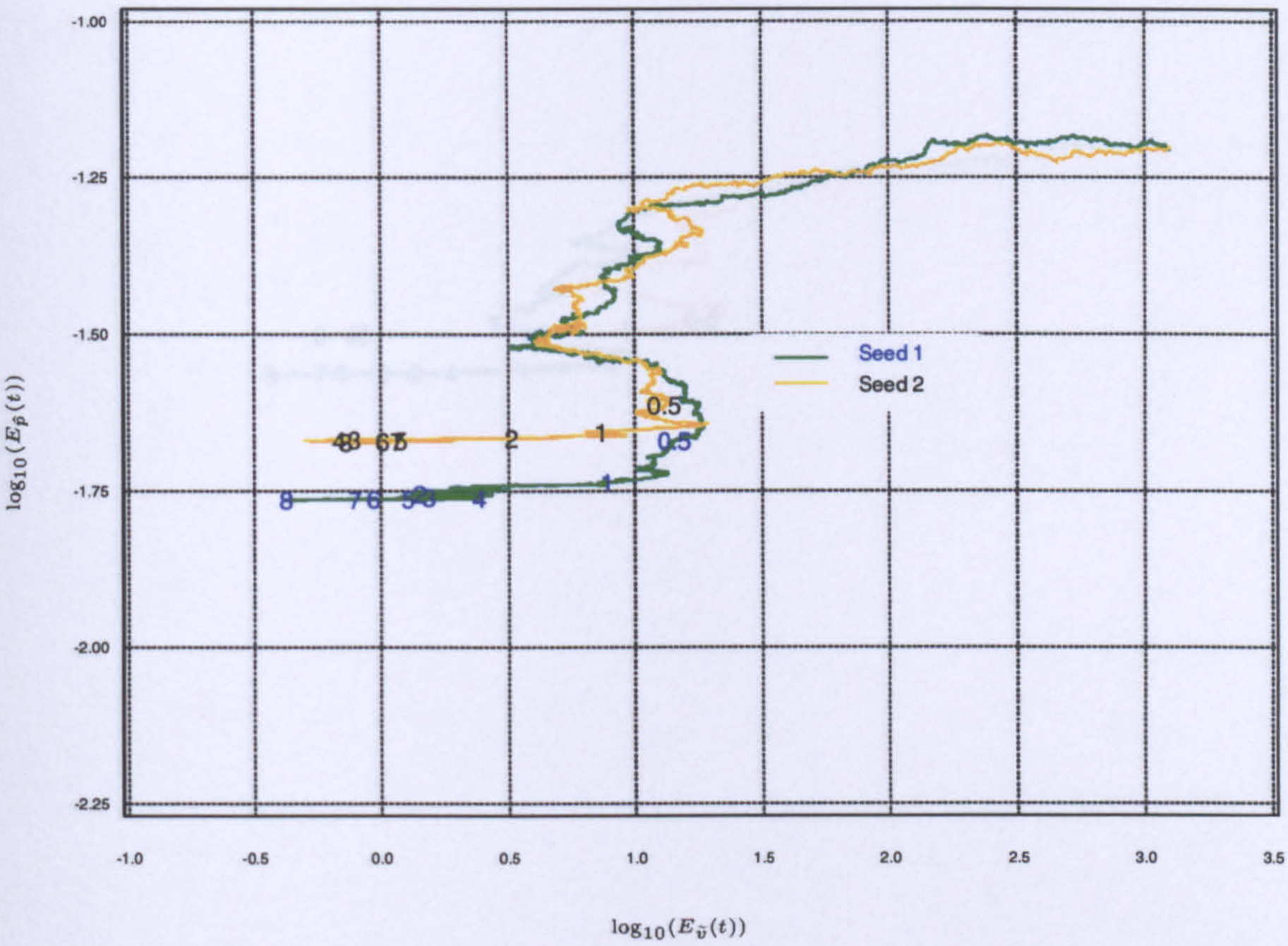


Figure 3.20: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; where $\mu = 5000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$. Twenty different seeds were used in all. We present results taken from two seeds to give us a good illustration of the method's performance. Tick marks were plotted for each of the two seeds at 5000, 10000 and every 10000 iterations thereafter, labelled 0.5, 1, 2, 3, 4, 5, 6, 7 and 8.

3.11 Running the optimiser concurrently with the P-learner - The evolution of the current value function and model estimates

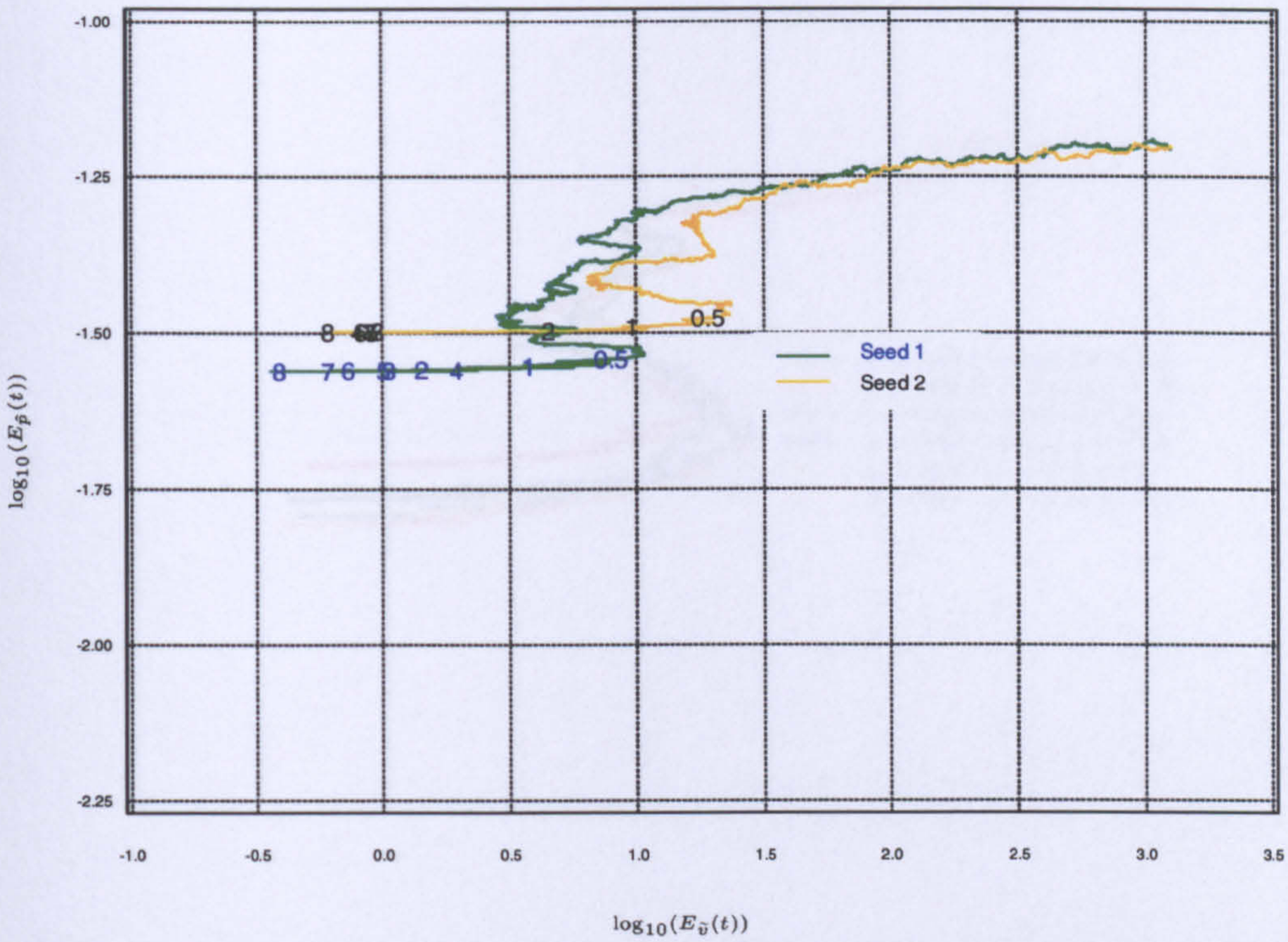


Figure 3.21: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; where $\mu = 5000.0$, $\sigma = 1000.0$, $\Delta = 0.01$ and $T = 80000$. Twenty different seeds were used in all. We present results taken from two seeds to give us a good illustration of the method's performance. Tick marks were plotted for each of the two seeds at 5000, 10000 and every 10000 iterations thereafter, labelled 0.5, 1, 2, 3, 4, 5, 6, 7 and 8.

3.11 Running the optimiser concurrently with the P-learner - The evolution of the current value function and model estimates

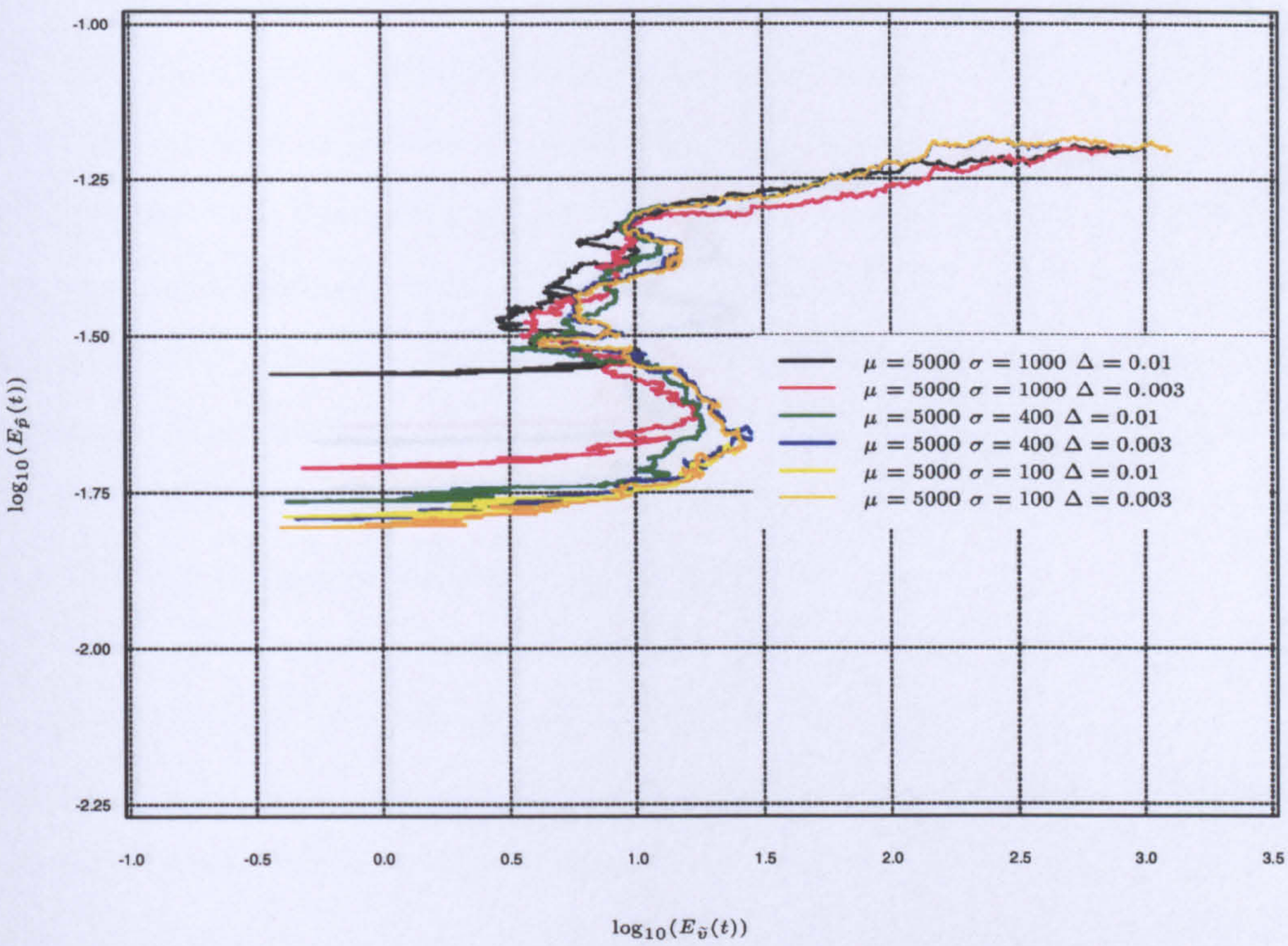


Figure 3.22: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for different values of $\mu = 5000$, σ, and Δ with $T = 80000$. The graph illustrates results taken from seed 1.

3.11 Running the optimiser concurrently with the P-learner - The evolution of the current value function and model estimates

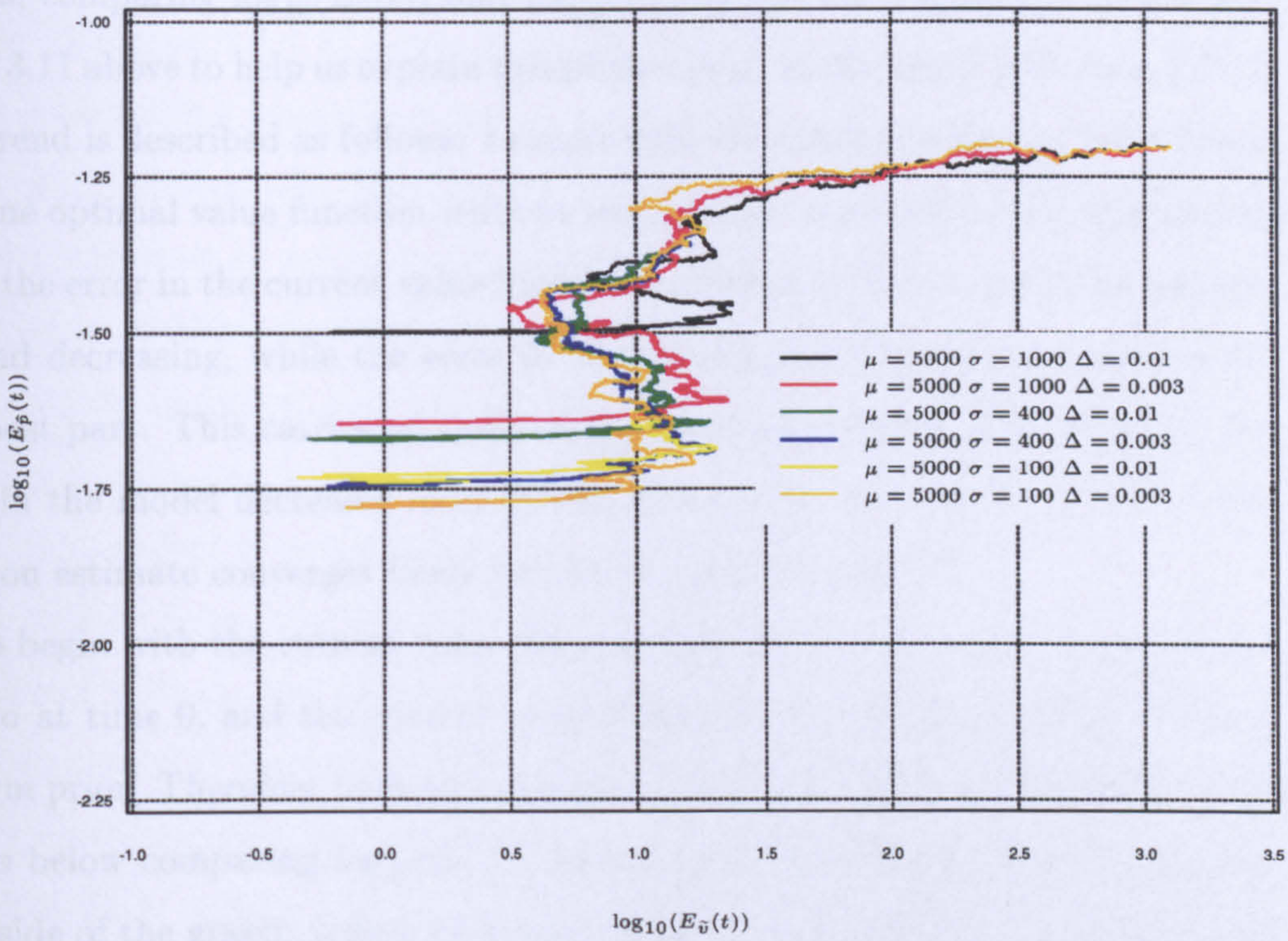


Figure 3.23: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for different values of $\mu = 5000$, σ, and Δ with $T = 80000$. The graph illustrates results taken from seed 2.

3.11.4 Consistent Trends

In this section we investigate the phenomena of consistent trend in the above graphs, comparing $\log_{10}(E_{\tilde{v}}(t))$ and $\log_{10}(E_{\hat{p}}(t))$. We use Figure 3.24 below and Table 3.11 above to help us explain this phenomena, as discussed in Section 3.11.1. The trend is described as follows: to start with the method seems to learn about the true optimal value function without learning too much about the true model. Then the error in the current value function estimate oscillates, noticably increasing and decreasing, while the error in the current model estimate decreases for the most part. This carries on until at the point of discrimination (time μ) the error in the model decreases more gradually and the error in the current value function estimate converges fairly rapidly to zero until time T.

To begin with the current value function estimates in all of the states are set to zero at time 0, and the current state transition probability estimates have a uniform prior. Therefore both sets of errors are all very large at time 0. Thus the figures below comparing $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ start at the top right hand side of the graphs before eventually making their way over to the left hand side.

At the start of each run when the optimiser is run concurrently with the p-learner, the convergence of the current value function estimate is comparable to the case of deterministic convergence (the optimiser when run on its own). Thus the current value function estimate converged fairly rapidly to the wrong model (the present model) without learning too much about the true model. It is for this reason that from $t = 0$ to $t = 2000$ the error in the current value function estimate decreased fairly quickly relative to the errors in the current model estimate. This is why once the current value function estimate had converged to the wrong model, the error in the current value function estimate did not change very much because the optimiser relied heavily on the information given to it by the p-learner.

3.11 Running the optimiser concurrently with the p-learner - The evolution of the current value function and model estimates

We know from time 0 to a time approaching μ the p-learner tries to sample all the actions in each state, which on occasions has the effect of making the method think a suboptimal policy is the true optimal one. We can see this by looking at Figure 3.4 - to begin with the optimiser had trouble differentiating between choosing a suboptimal policy and the true optimal policy. This caused errors in the current value function estimate to increase and sometimes decrease, due to compensating errors in the difference between the true state-transition probabilities and the current state-transition probability estimates at time t. Not only that, the optimiser only updates the current value function estimate at one state per update and both the optimiser and the p-learner visit their own set of states independent of one another, so it may take the optimiser a while for it to realise that its behaviour is sometimes suboptimal.

We can look at two particular cases in Figure 3.24 below. In Figure 3.24 below we note that from $t = 3000$ to 6000 the error in the current value function estimate increased fairly rapidly with respect to time and after $t = 30000$ the error decreased fairly rapidly with respect to time. If we look at Table 3.11 above we can see that the time intervals from $t = 3060$ to 3598 and from $t = 3649$ to 6043, were the time periods when the optimiser lost the true optimal policy in favour of a suboptimal policy. Therefore in these instances the optimiser converged to a suboptimal model based on the information given to it by the p-learner. This corresponds to the behaviour of the method's performance from $t = 3000$ to 6000 shown in Figure 3.24 below. The period from $t = 3000$ to 6000 is the period with the largest error in the current value function estimate (apart from that at the beginning of the run).

The results quoted in Table 3.11 tell us the current value function estimate converged after 8909 iterations. However, the p-learner still sampled suboptimal actions until some time after time μ (see Figure 3.6). Thus we see for a short time period, after $\mu = 20000$, the error increased, that is until the p-learner sampled the important actions in each state enough times so that the error in the current

value function estimate eventually converged to zero. Even at $T = 80000$ we see that there is still enough sampling error in the current state-transition probability estimates for us not to see $\log(E_{\tilde{v}}(t)) = -6$, but from $t = 40000$ onwards the tick marks show a vast improvement in $E_{\tilde{v}}(t)$

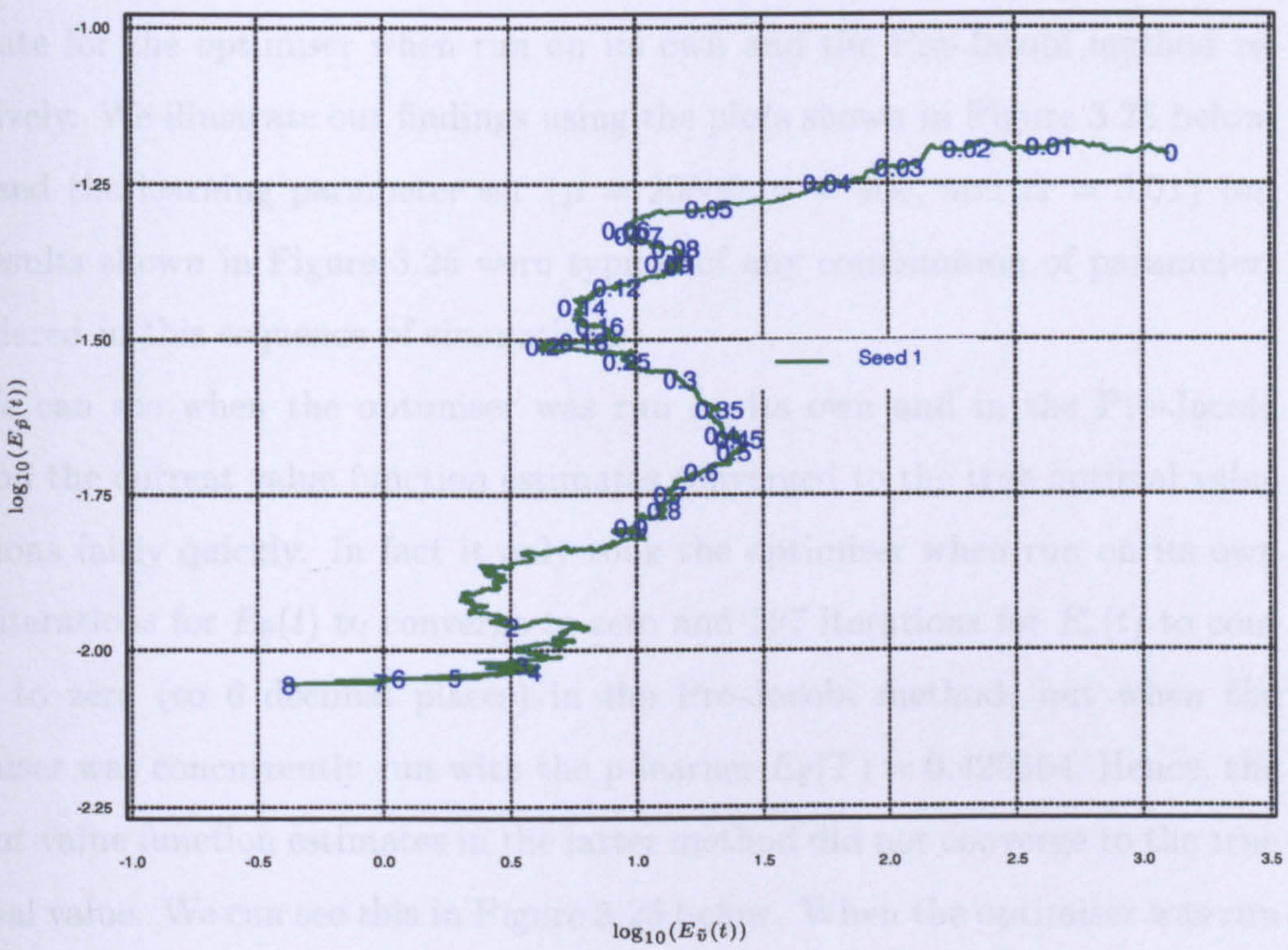


Figure 3.24: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$ for seed 1; where $\mu = 20000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$. The plot shows us the method's performance at a wide range of time points. Tick marks are labelled from 0.01 to 8 where tick mark 1 represents the method's performance at step time 10000.

3.11.5 Comparing different methods

To compare the performance of the optimiser when run concurrently with the p-learner we compared it with the performance of the optimiser when run on its own and the Pre-Jacobi method, as discussed in Section 3.5.3.3. We compared the error in the current value function estimates for each method using Equation (3.11.5) with step time t. Firstly, we substituted $\widehat{v}$ and v for $\widetilde{v}$ in Equation (3.11.5) to represent the root mean square error of the current value function estimate for the optimiser when run on its own and the Pre-Jacobi method respectively. We illustrate our findings using the plots shown in Figure 3.25 below. We used the learning parameter set $\{\mu = 20000, \sigma = 400, \text{ and } \Delta = 0.01\}$ but the results shown in Figure 3.25 were typical of any combination of parameters considered in this sequence of simulations.

We can see when the optimiser was run on its own and in the Pre-Jacobi method the current value function estimates converged to the true optimal value functions fairly quickly. In fact it only took the optimiser when run on its own 2214 iterations for $E_{\widehat{v}}(t)$ to converge to zero and 207 iterations for $E_v(t)$ to converge to zero (to 6 decimal places) in the Pre-Jacobi method, but when the optimiser was concurrently run with the p-learner $E_{\widetilde{v}}(T) = 0.420564$. Hence, the current value function estimates in the latter method did not converge to the true optimal value. We can see this in Figure 3.25 below. When the optimiser was run on its own and in the Pre-Jacobi method the errors in the current value function estimates decreased monotonically until convergence. However, when the optimiser was run concurrently with the p-learner the errors decreased for the most part, but Figure 3.25 shows that the errors occasionally increased. This is due the fact the optimiser depends heavily on the estimates of the true state transition probabilities given to it by the p-learner. An example of this is discussed in Section 3.11.4 above.

3.11 Running the optimiser concurrently with the p-learner - The evolution of the current value function and model estimates

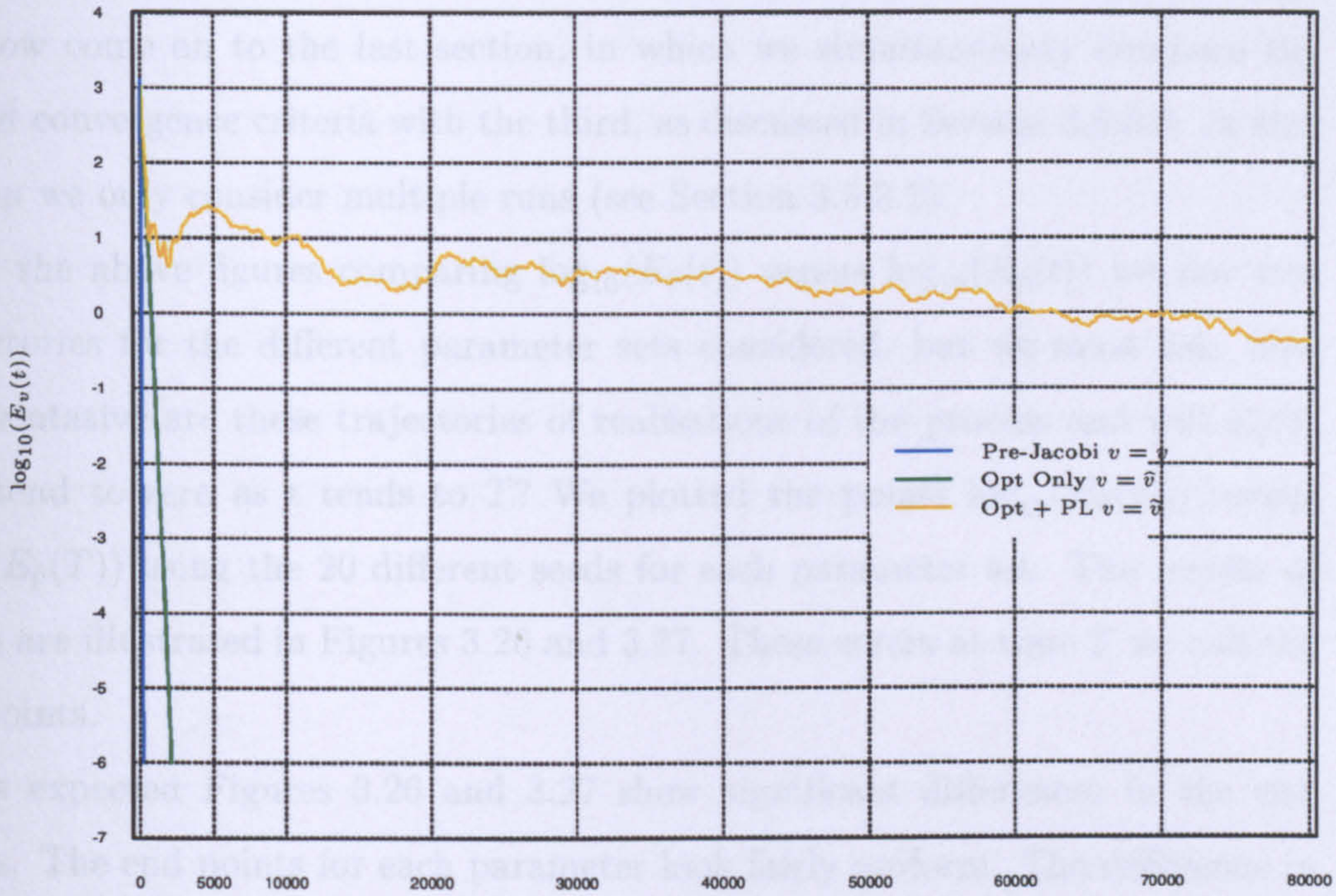


Figure 3.25: The graph above is a plot of root mean square error of current value function estimate at $\log_{10}$ against step time t using parameters $\mu = 20000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ for seed 1. We wanted to compare the error in the value function for the 3 different methods; the optimiser run on its own (Opt Only), the optimiser run concurrently with the p-learner (Opt + PL) and the conventional Pre-Jacobi method. We plotted the errors in the value function for these particular parameters because qualitatively speaking they display roughly the same typical behaviour over the 3 different methods as in any other set of parameters.

3.12 The optimiser run concurrently with the p-learner - End points

We now come on to the last section, in which we simultaneously compare the second convergence criteria with the third, as discussed in Section 3.5.3.3. In this section we only consider multiple runs (see Section 3.5.3.1).

In the above figures comparing $\log_{10}(E_{\tilde{v}}(t))$ versus $\log_{10}(E_{\hat{p}}(t))$ we saw two trajectories for the different parameter sets considered, but we must ask, how representative are these trajectories of realisations of the process and will $E_{\tilde{v}}(t)$ ever tend to zero as t tends to T? We plotted the points $\log_{10}(E_{\tilde{v}}(T))$ versus $\log_{10}(E_{\hat{p}}(T))$ using the 20 different seeds for each parameter set. The results of which are illustrated in Figures 3.26 and 3.27. These errors at time T we call the end points.

As expected Figures 3.26 and 3.27 show significant differences in the end points. The end points for each parameter look fairly uniform. The difference in the end points is caused by the stochasticity due to the way actions are chosen using different seeds and different parameter sets. This happens when the function $\gamma'(t)$ becomes appreciable (rises above zero). The function $\gamma'(t)$ becomes appreciable near time μ if σ is small and time $t << \mu$ if σ is large.

For $\mu = 5000$ the end points (for the 20 different seeds) are more clustered compared with the $\mu = 20000$ and 10000 cases. We recall the results found using the learning parameter set $\{\mu = 5000, \sigma = 1000, \Delta = 0.01\}$ shown in Table 3.10 above, we see that the current action estimates only converged to the optimal action in state 8, 13 times out of 20, and if we look at Figure 3.27 the majority of its end points have large errors in the value function compared with most. This is caused by the discrepancies in the estimated model compared with the true model. As for the other parameter sets for $\mu = 5000$ the eventual action estimate in state 8 was the true optimal action for most seeds therefore their cluster of

end points do not look too dissimilar to the cluster of end points for $\mu = 20000$ and 10000.

For $\mu = 20000$ and 10000 (see Figure 3.26) we see 9 outliers in the left hand tail of the overall cluster. Table 3.10 tells us that in the $\mu = 20000$ and 10000 case the current action estimate converged to the true optimal action in state 8 most of the time. Thus we would expect the value function estimates at time T to be closer to the optimal value function for the $\mu = 20000$ and 10000 cases than for the $\mu = 5000$ case. But $E_{\tilde{v}}(T)$ does not equal zero in these cases. This makes us wonder whether we are getting to the limit when we try to estimate the state-transition probabilities and if noise in the current value function estimates will always be present.

3.12 The optimiser run concurrently with the p-learner - End points

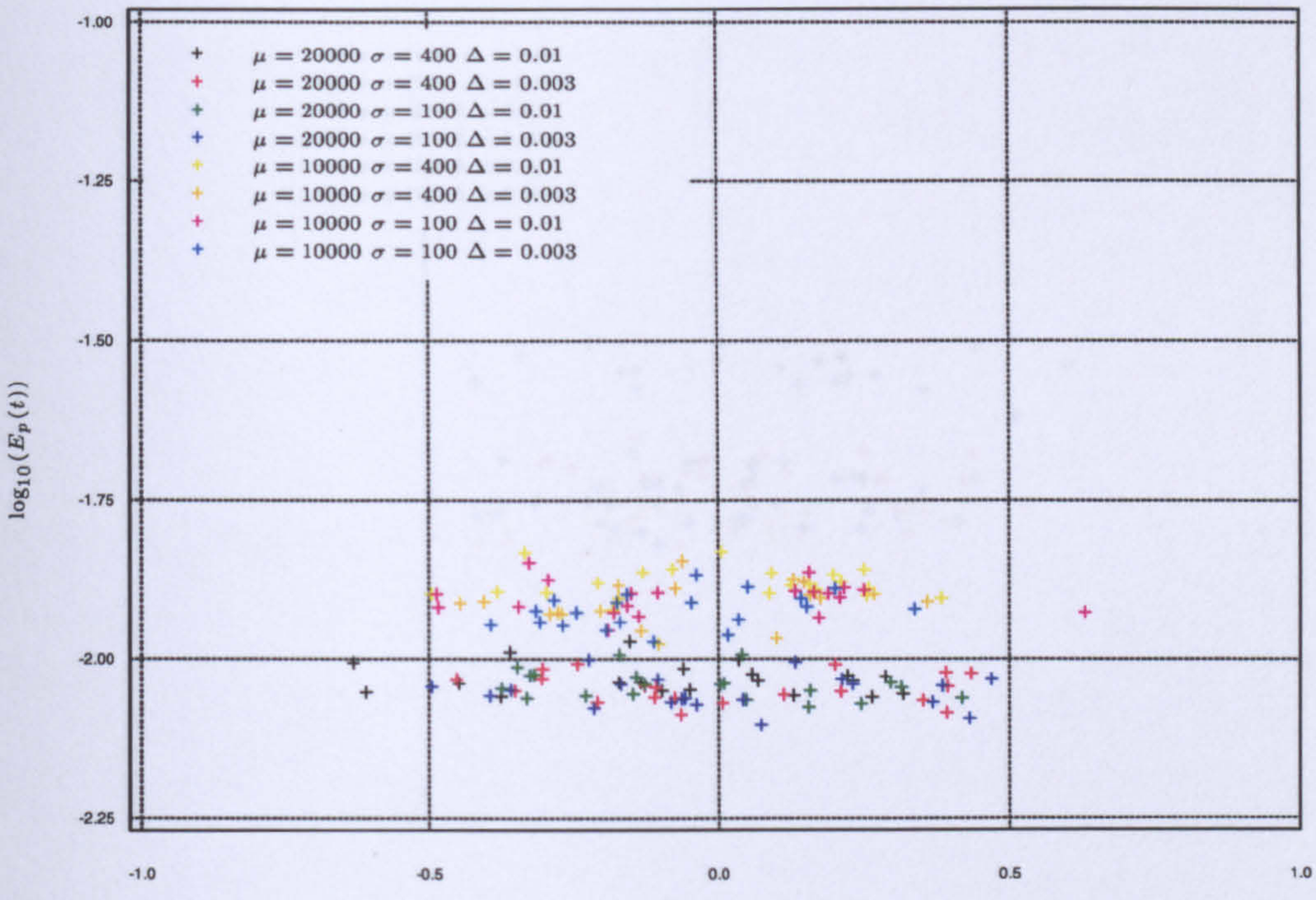


Figure 3.26: The graph above is a plot of $\log_{10}(E_v(t))$ against $\log_{10}(E_p(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for the different combinations of $\mu = 10000, 20000.0$, $\sigma = 100.0, 400.0$, and $\Delta = 0.003, 0.01$. The graph illustrates the position of the end points of all 20 simulations using different seeds for each parameter set.

3.12 The optimiser run concurrently with the p-learner - End points

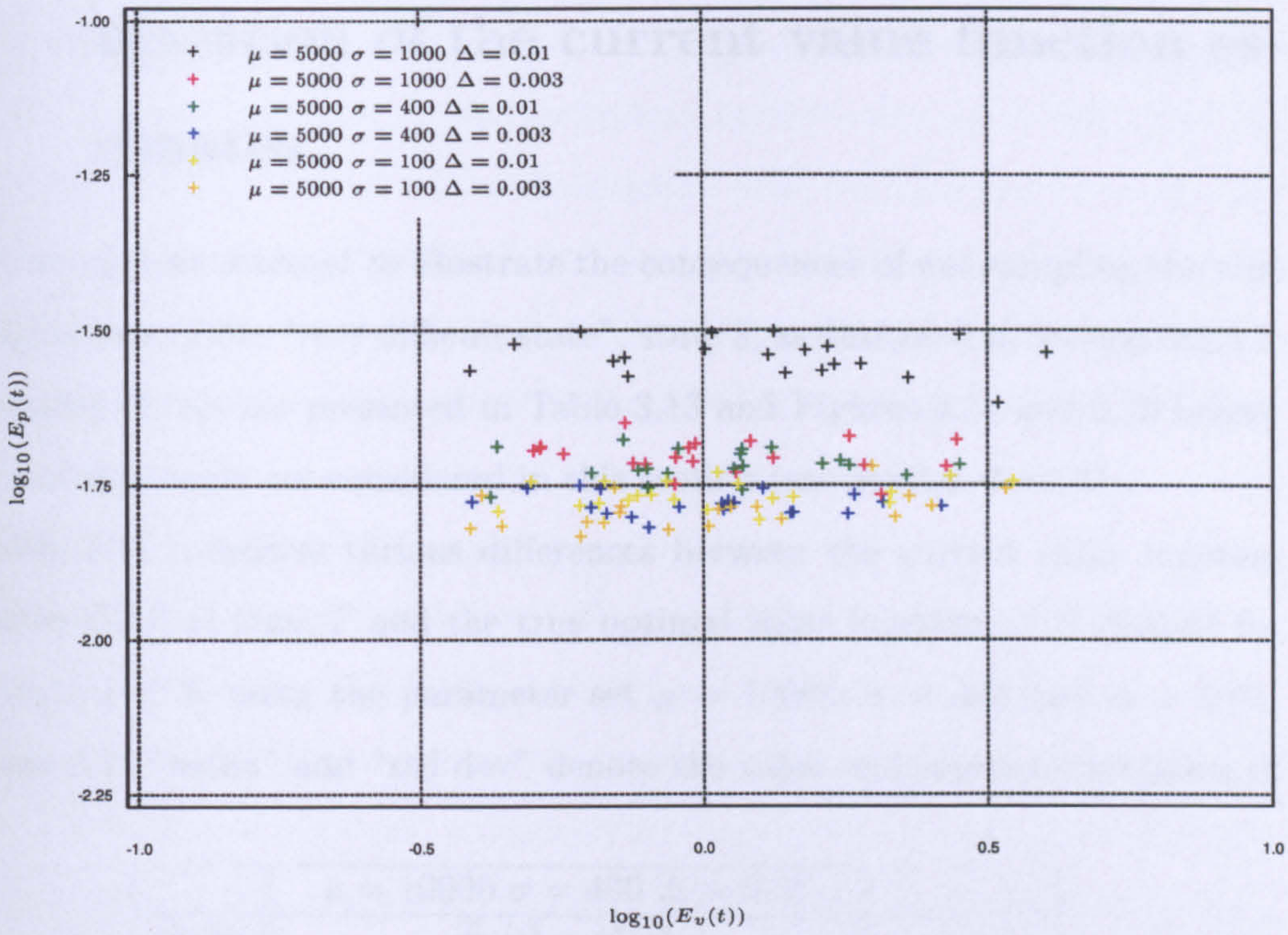


Figure 3.27: The graph above is a plot of $\log_{10}(E_v(t))$ against $\log_{10}(E_p(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for the different combinations of $\mu = 10000, 20000.0$, $\sigma = 100.0, 400.0$, and $\Delta = 0.003, 0.01$. The graph illustrates the position of the end points of all 20 simulations using different seeds for each parameter set.

3.13 The optimiser run concurrently with the p-learner - The average bias and standard deviation of the current value function estimates

In this section we attempt to illustrate the consequences of not sampling the true optimal action in the "very difficult state", state 8, as discussed in Section 3.5.3.3. The results of this are presented in Table 3.13 and Figures 3.28 and 3.29 below. Only multiple seeds are considered in this section (see Section 3.5.3.1).

Table 3.13 compares various differences between the current value function estimates $\widetilde{v}_T(i)$ at time T and the true optimal value function $v^*(i)$ defined for each state $i \in S$, using the parameter set $\mu = 10000$, $\sigma = 400$ and $\Delta = 0.01$. In Table 3.13 "mean" and "std dev" denote the value and standard deviation of

	$\mu = 10000\ \sigma = 400\ \Delta = 0.01$				
State	$\widetilde{v}_T(i) - v^*(i)$				
$i \in S$	mean	min	max	std dev.	$v^*(i)$
0	-0.279	-2.038	2.462	1.338	1341.166
1	-0.054	-1.968	2.412	1.149	1318.535
2	-0.124	-1.724	1.423	1.008	1229.388
3	-0.215	-2.138	2.120	1.109	1230.372
4	-0.176	-2.901	2.662	1.330	1254.341
5	-0.174	-2.177	2.407	1.321	1242.126
6	-0.099	-2.520	2.097	1.258	1212.976
7	-0.190	-1.828	2.760	1.273	1342.630
8	0.263	-2.154	4.448	1.947	1225.870
9	0.064	-1.523	2.404	1.248	1213.414

Table 3.13: The table above displays the mean (averaged over the 20 simulations), the minimum value, the maximum value and the standard deviation (averaged over the 20 simulations) of $\widetilde{v}_T(i) - v^*(i)$ for each state $i \in S$. The parameters used were $\mu = 10000\ \sigma = 400$, $\Delta = 0.01$ and $T = 80000$.

3.13 The optimiser run concurrently with the p-learner - The average bias and standard deviation of the current value function estimates

$\widetilde{v}_T(i) - v^*(i)$ averaged over the 20 simulations. Throughout this section we refer to "mean" and "std dev" as average bias and average standard deviation of $\widetilde{v}_T(i)$ respectively. The positive bias in state 8 is typical when the p-learner has trouble estimating the true state-transition probabilities in that state. Figures 3.28 and 3.29, on the other hand, are graphical displays summarising the average bias and the average standard deviation of the current value function estimates at time T at each $i \in S$ and for all the 8 sets of parameters used in our sequence of simulations. In each figure observations for each parameter set are labelled in one colour. Observations for each state are labelled from 0 to 9. We plotted these figures because they convey more information than would be the case if we were to present one table for each parameter set.

As expected, looking at Figures 3.28 and 3.29 we see the principal features are the observations for state 8. If we look back to Table 3.10, the table tells us which parameter sets had trouble sampling the true optimal action in state 8. In Table 3.10 we notice the worst parameter sets for $\mu = 20000$ and 10000 were the ones corresponding to $\sigma = 400$, $\Delta = 0.003$, and $\sigma = 400$, $\Delta = 0.01$, respectively. Figure 3.28 shows that the two observations corresponding to these parameter sets in state 8 are the two right most observations in the picture. In fact, 4 out of 8 observations corresponding to state 8 are to the far right of the picture.

In Figure 3.28 most of the biases in the different states are negative, except for states 8 and 9, which is a significant result. We note the majority of observations for state 9 in Figure 3.28 have a bias around zero. We previously saw in Tables 3.6, 3.7, 3.8 and 3.9 above that state 9 is the state that was visited most often. However, this does not tell anything about the bias, only the standard deviation. In fact most the time state 9 had one of the smallest standard deviations in comparison to the rest of the states in its parameter set.

Figure 3.25 (comparing the three different methods) demonstrates that if we know the true state-transition probabilities the optimal value function will converge fairly quickly. So positive bias cannot be a result of not visiting states

often enough. It is because we do not have good enough estimates of the true state-transition probabilities, especially in state 8 - and because the current value function estimates in state 8 are far from the final solution, the estimates of the optimal value function in the other states will not converge because they consist of the estimates for state 8. We note the positions of the observations for different learning parameters in states 8. For all states other than state 8, a similar pattern of observation clusters is apparent in all parameter sets.

In Figure 3.29 on the other hand, all of the observations corresponding to state 8 are on the far right of the picture. Table 3.10 confirms this result. We note from Table 3.10 that, in 20 simulations, using the learning parameter set $\{\mu = 5000, \sigma = 1000, \Delta = 0.01\}$, the p-learner only chose the true optimal action in state 8, 13 times over the best suboptimal action. Hence the optimiser only chose the true optimal policy 13 times out of 20. Consequently, the standard deviation and bias corresponding to state 8 is extremely large compared with the rest. A similar argument can be said using the learning parameter $\{\mu = 5000, \sigma = 100, \Delta = 0.01\}$. We can also see from Figure 3.29 that if the observations for state 8 have large positive biases, the other observations corresponding to their parameter set also have positive biases.

In conclusion, evidence suggests that even if the overall costs between the true optimal action and the best suboptimal action in a particular state are fairly close, sampling the best suboptimal action over the true optimal action for long periods may result in serious consequences especially when it comes to estimating the optimal value function. We have shown that using wrong estimates of the true state-transition probabilities in the current value function estimate in a particular state will make the solution converge to a suboptimal value function. This result was less apparent when we looked only at the two realisations comparing $\log_{10}(E_v(T))$ with $\log_{10}(E_p(T))$ in the previous sections.

3.13 The optimiser run concurrently with the p-learner - The average bias and standard deviation of the current value function estimates

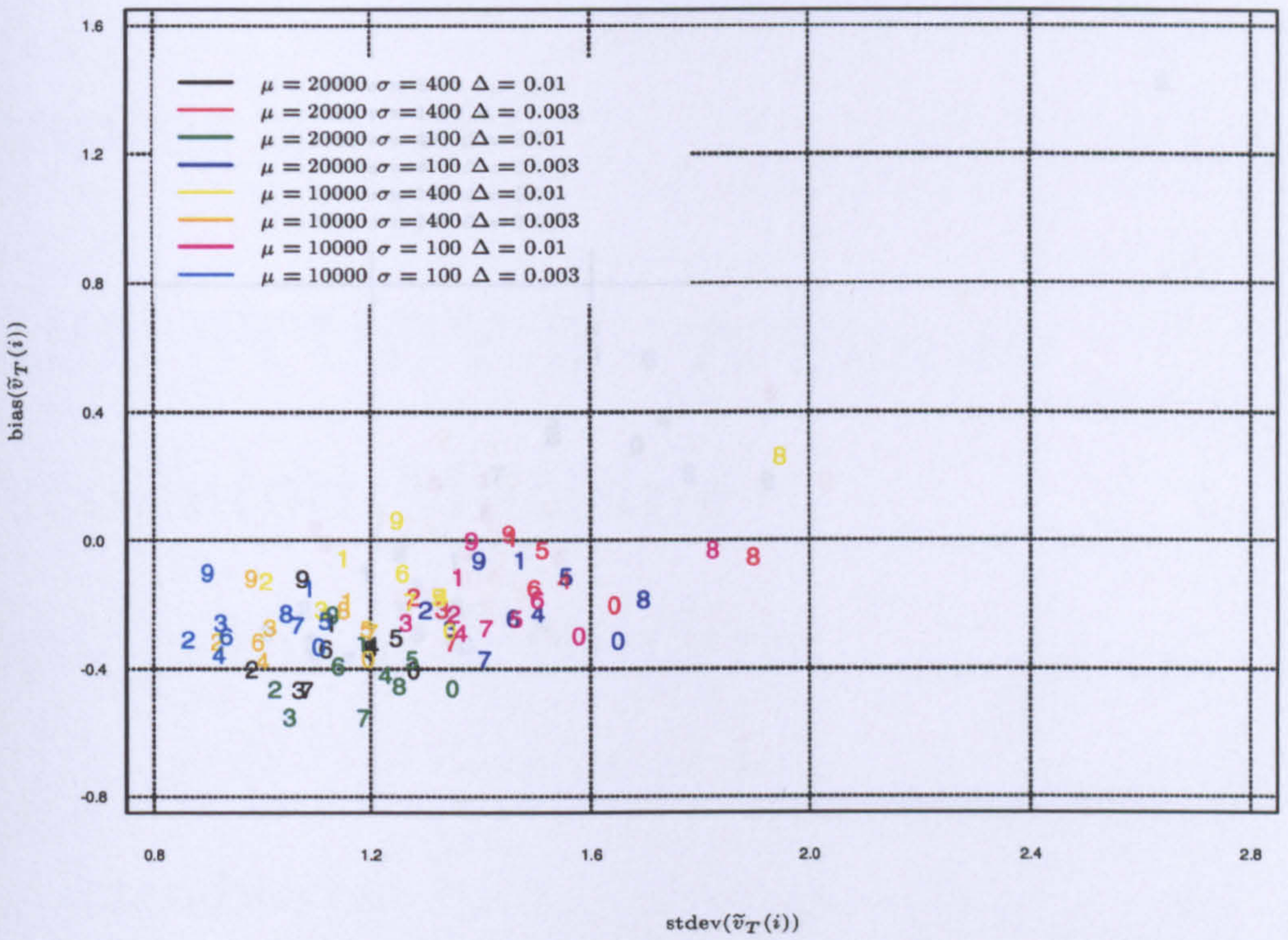


Figure 3.28: The graph above is a plot of the average standard deviation of $\widetilde{v}_T(i)$ against the average bias of $\widetilde{v}_T(i)$ defined for all states $i \in S$ at time T for the parameter sets considered for $\mu = 20000$ and $\mu = 10000$. Each observation consists of results taken from the 20 simulations for each parameter set. We used one colour for each parameter set and each observation represents the value of each state.

3.13 The optimiser run concurrently with the p-learner - The average bias and standard deviation of the current value function estimates

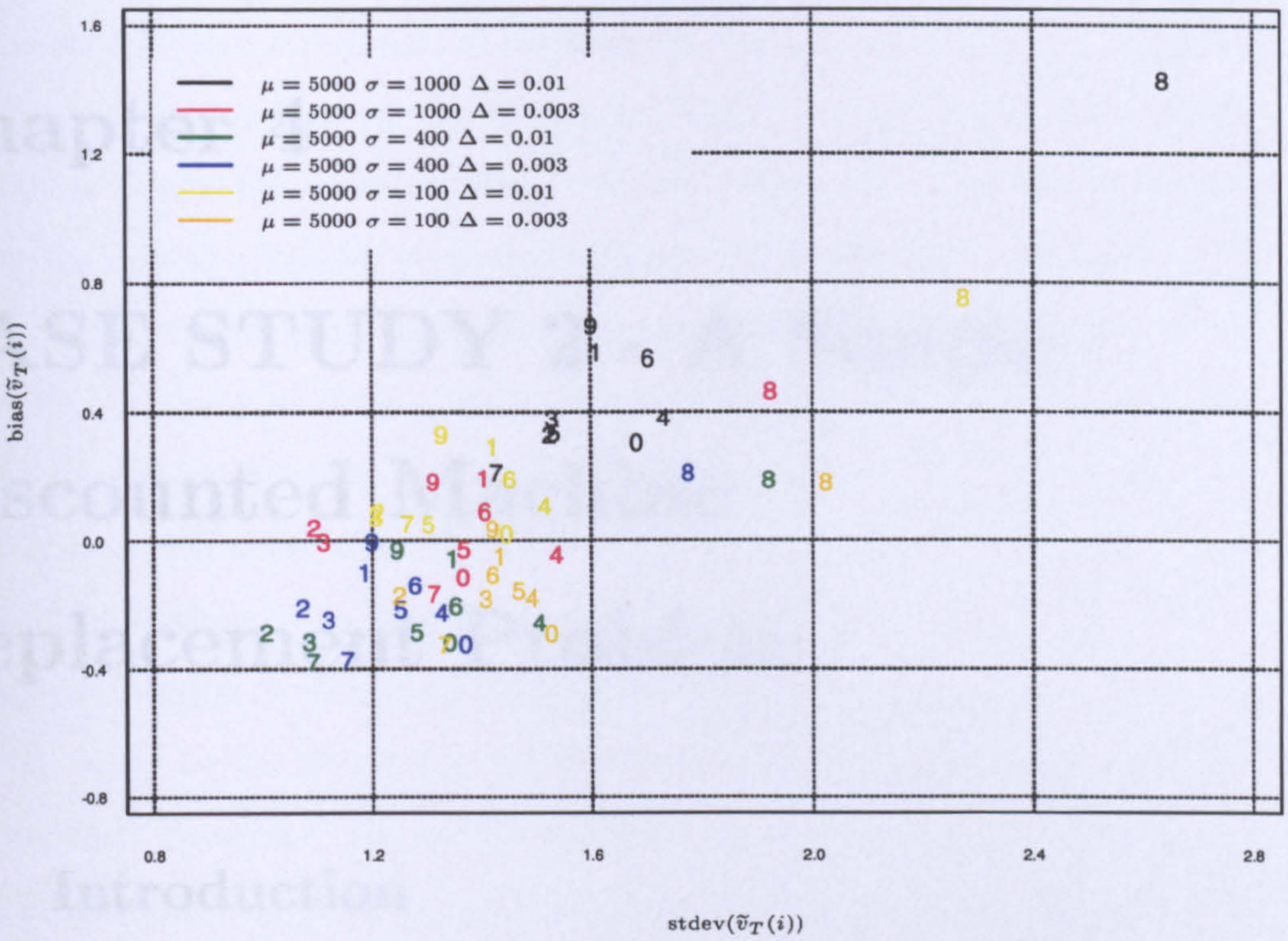


Figure 3.29: The graph above is a plot of the average standard deviation of $\widetilde{v}_T(i)$ against the average bias of $\widetilde{v}_T(i)$ defined for all states $i \in S$ at time T for all the parameter sets considered for $\mu = 5000$. Each observation consists of results taken from the 20 the simulations for each parameter set. We used one colour for each parameter set and each observation represents the value of each state.

Chapter 4

CASE STUDY 2 - A Simple Discounted Machine Replacement Problem

4.1 Introduction

This chapter considers a simple discounted machine replacement (sparsely connected) problem. In the previous chapter we investigated the performance of our methods for different learning parameter sets for a "fully connected" system. We compared the results of this analysis with results generated by the more conventional Pre-Jacobi method. We also showed that the computational cost we pay for not knowing the true state transition probabilities is fairly high.

In this chapter we look at modifications to the previously discussed methodology which help us deal with more "sparsely connected" systems. We call this previously discussed methodology a *standard* approach. We use the same criteria as in the preceding chapter to evaluate the ability of the modified methods to find a true optimal policy and the true optimal value function in each state $i \in S$. In all instances a fixed simulation horizon budget of size T was used.

4.1 Introduction

In this chapter we will look at three different methodologies, hence the chapter is split into three main parts. In the first part we will see that applying our *standard* version to this problem has a number of difficulties associated with it. For these reasons, in the second part of the chapter, we extend the *standard* approach using an indexed method in the optimiser which we call the *indexed* version. However, problems still remain in the p-learner. Therefore, in the last part of the chapter we look at a *coupled* version where we couple both the *indexed* optimiser and p-learner, rather than running them independently. Although not widely applicable, the *coupled* version gives us an interesting insight into the bounds of behaviour of the system, given that sampling problems in the p-learner can be eradicated. All these algorithms are compared in turn with the conventional Pre-Jacobi method.

We first describe the illustrative problem. After reviewing the *standard* methodology we apply to the illustrative problem, we discuss the function of the algorithm at each iteration. The choice of learning parameters for choosing actions for all algorithms used in this chapter is then discussed. We discuss the different output reported for each algorithm. We then outline various the criteria which we use to analyse the performance of the *standard* algorithm across a different range of parameters, before applying it to the particular problem analysed in this case study.

After discussing the problems encountered with the *standard* methodology used in the previous chapter, we describe an improved performance algorithm called the *indexed* version which helps us deal more effectively with sparsely connected systems. We describe its mechanics at each iteration when system parameters are both known and unknown, so as to precisely define the criteria we will use to evaluate the success of our algorithms. We outline the various criteria we use to evaluate its performance and then discuss the outcome of our simulations. We then present the *coupled* version which is a way of overcoming the problems experienced in the p-learner. We discuss in detail the performance

of the algorithm at each iteration and we close by presenting some results. We end the chapter comparing the performance of the *indexing* method with that of the *standard* method using the problem illustrated in the previous chapter.

4.2 Problem Description

A simple infinite horizon machine replacement problem with a discount factor α was set up to illustrate how robust the methods described in Section 3.3 are. This case study is representative of a problem studied on Page 248 of Puterman (1994). The problem described below helped motivate the development of our improved performance algorithms using modifications of an index method, described in Section 4.7. Throughout this chapter we used the problem to compare the performance of our algorithms with the more conventional Pre-Jacobi method.

A machine is used in manufacturing and at each time t, a decision has to be made whether to replace it with a new one. Machines deteriorate over time and their condition is categorised into states $i \in S$ where $S = \{0, 1, 2, \ldots n\}$. The lower the value of the state the better its condition. At each time t, in states 0 through to $n-1$, the controller has a choice of two actions, 0 and 1. Action 0 denotes the action of replacing the machine for a new one, and action 1 denotes the action of continuing with the present machine until the next decision epoch. Note that in state n only action 0 is available since the machine is in its worst conceivable condition, and it must be replaced.

Let i_t be the state at time t. If $i_t = i$ at time t and the decision is to "replace", we incur a cost $c_i(0)$ and a new machine is put in position ready to start at time $t+1$ where,

$$i_{t+1} = \begin{cases} 0 & \text{wp} \quad p_0, \\ 1 & \text{wp} \quad 1-p_0 \quad \text{and}, \\ j & \text{wp} \quad 0 \quad \text{if} \quad j \neq 0 \text{ or } 1. \end{cases} \tag{4.2.1}$$

However, if $i_t = i$ at time t and the decision is "do not replace", a maintenance cost $c_i(1)$ is incurred where,

$$i_{t+1} = \begin{cases} i & \text{wp} \quad p_1, \\ i+1 & \text{wp} \quad 1-p_1 \quad \text{and}, \\ j & \text{wp} \quad 0 \quad \text{if} \quad j \neq i \text{ or } i+1. \end{cases} \tag{4.2.2}$$

Costs are discounted over an infinite horizon time at rate α.

In this chapter all of the numerical results are based on the following numerical parameters:

- $S = \{0, 1, \ldots, 11\}$,
- $U = \{0, 1\}$ with only action 0 available in state 11,
- $p_0 = 0.38 \quad 1 - p_0 = 0.62$,
- $p_1 = 0.41 \quad 1 - p_1 = 0.59$,
- $c_i(0) = 10.2 \quad c_i(1) = i$,
- discount factor $\alpha = 0.75$.

Contrary to the "fully connected" system studied in Chapter 3, the machine replacement is a "sparsely connected" system because it is not possible to move from a state $i \in S$ to each state $j \in S$ given any action $u \in U(i)$ in one time step. It is for this reason that the system parameters such as the immediate costs and the state transition probabilities were carefully chosen. Owing to the structure of the problem we chose the state transition probabilities in such a way as to make it easier for the controller to visit the higher valued states. In later sections we will see that sampling the higher valued states in the optimiser and p-learner is not as easy as first we anticipated. In conjunction with the state transition probabilities, the immediate costs were chosen to exemplify varied behaviour in the system for different states. Twelve states in the state space S were chosen because we knew

that we only had a finite simulation horizon of size T. The bigger the state space, the shorter the amount time in which to sample each individual state. We chose a large value for α for the same reason as that stated in Section 3.2.

It is known that an optimal stationary policy exists (Puterman 1994) for the underlying infinite discounted MDP which is a threshold policy with threshold (state) I of the form,

$$\pi(i) = \begin{cases} 0 & \text{if} \quad i \geq I \quad \text{and,} \\ 1 & \text{if} \quad i < I, \end{cases} \tag{4.2.3}$$

with a corresponding value function,

$$\begin{aligned} v(i) &= c_i(\pi(i)) + \alpha \sum_{j \in S} p_{ij}(\pi(i)) v(j), \\ &= \alpha \{0.41 v(i) + 0.59 v(i+1)\} \quad \text{for} \quad 0 \leq i < I, \\ &= \alpha \{0.38 v(0) + 0.62 v(1)\} \qquad \text{for} \quad i \geq I. \end{aligned} \tag{4.2.4}$$

Since we know there exists an optimal stationary policy, the optimal "Q-values" show it in Table 4.1 below.

Table 4.1 below is a table of the optimal Q-values $Q^*(i,u)$ and the optimal actions $\pi^*(i)$, defined for each state $i \in S$ and action $u \in U(i)$. We can see that the value iteration method used to obtain the results presented in Table 4.1 decomposed the state space under the stationary policy into two distinct classes: a recurrent class and a transient class, the recurrent class being states $\{0,1,2,3,4\}$ and the transient class being states $\{5,6,7,8,9,10,11\}$. A state is recurrent if a return to it is certain to occur eventually, whereas a state is transient if a return to it is not certain.

We will see in Section 4.6 that when we apply the *standard* optimiser and p-learner to this "sparsely connected" problem, difficulties occur when we try to visit and observe the higher valued states both in the optimiser and the p-learner respectively. This problem was not present in the "fully connected" case,

described in the previous chapter. In addition to this Table 4.1 shows that in state 4 it is not a clear cut decision which action is best, even when the system parameters are known. However, this is similar to the case in state 8 in the previous chapter.

State	$Q^*(i,u)$		$\pi^*(i)$
$i \in S$	$u=0$	1	
0	16.196	**5.921**	1
1	16.196	**9.265**	1
2	16.196	**12.240**	1
3	16.196	**14.636**	1
4	16.196	**16.125**	1
5	**16.196**	17.147	0
6	**16.196**	18.147	0
7	**16.196**	19.147	0
8	**16.196**	20.147	0
9	**16.196**	21.147	0
10	**16.196**	22.147	0
11	**16.196**	NA	0

Table 4.1: The table above presents the optimal Q-values, $Q^*(i,u)$ for this particular problem defined for each state $i \in S$ and action $u \in U(i)$, and the corresponding optimal actions $\pi^*(i)$ defined for each state $i \in S$. These values were obtained using value iteration. Note, all of the bold figures in the above table correspond to the true optimal value function $v^*(i)$ defined for each state $i \in S$.

4.3 The Methodology Used In The First Part Of This Chapter

4.3.1 Review

This is the first part of the chapter which discusses the *standard* version of the methodology used in previous chapter. In this section we briefly re-cap on the methodology before we apply it to the problem described in Section 4.2 above.

We discuss the two extensions of the *standard* version, the *indexed* and *coupled* version, in the second and third parts of the chapter respectively.

In all of the algorithms described in this chapter, it is the optimiser that stochastically approximates the optimal value function $v^*(\cdot)$ and the optimal policy $\pi^*(\cdot)$ and it is the p-learner that updates the current estimates of the state transition probabilities (see Section 3.3.1). Note the p-learner is only used in conjunction with the optimiser when the true state transition probabilities are unknown. Thus, when the system parameters are known the optimiser is run on its own (the known P case) and when they are unknown the optimiser is run concurrently with the p-learner (the unknown P case). In both cases the other system parameters, the immediate costs, are known. When the optimiser is run concurrently with the p-learner we first iterate the optimiser and then the p-learner.

4.3.2 The Standard Version

The *standard* version consists of exactly the same procedures as those described in Section 3.3, with the exception of a slight modification to the way the p-learner updates the current estimates of the state transition probabilities. Throughout this chapter we make it a little easier for the p-learner to estimate the true state transition probabilities, because we are assuming we understand the nature of problem. Since we know that certain transitions to a state j given a state $i \in S$ and action $u \in U(i)$ are impossible, we inform the p-learner of this information at the start of each simulation. Thus the p-learner updates the current estimates of the state transition probabilities at time t where,

$$\hat{p}_{ij}^t(u) = \{x_{ij}^u(t)+1\}\left\{\sum_{j\in S} x_{ij}^u(t)+2\right\}^{-1} \quad \text{if} \quad p_{ij}(u)\neq 0 \quad \text{and,} \quad (4.3.5)$$

$$= 0 \quad \text{if} \quad p_{ij}(u)=0.$$

Thus Equation (4.3.5) above is used instead of,

$$\hat{p}_{ij}^{t}(u) = \{x_{ij}^{u}(t) + 1\} \left\{ \sum_{j \in S} x_{ij}^{u}(t) + |S| \right\}^{-1}, \tag{4.3.6}$$

as described in Section 2.3. In both equations $x_{ij}^{u}(t)$ denotes the observed number of transitions from state i to state j using action u before time t. Note that the only difference between Equations (4.3.5) and (4.3.6) above is that in Equation (4.3.5) we have taken into consideration the known structure of the problem. In the machine replacement problem, given we observe state $i \in S$ and we choose any available action $u \in U(i)$, the p-learner can only move to 2 other states, unlike in the "completely connected" system where it was possible to move to $|S|$ states.

4.4 Simulation Procedure

4.4.1 Choosing Actions - Choice of Parameters

Unless otherwise stated the results presented in this chapter are from algorithms which choose actions using method 3. Method 3 is reviewed in Section 3.3.4. The reasons why we used method 3 instead of methods 1 or 2 are the same as those stated in Section 3.4. We report results from the various combination of parameter sets $\{\mu, \sigma, \Delta\}$ using parameters $\mu = 5000, 10000, 20000$, $\sigma = 100, 400$ and $\Delta = 0.003, 0.01$ with $T = 80000$, as in Chapter 3. We re-iterate that μ denotes the amount of learning time, σ denotes the speed at which we move from uniform sampling to focused sampling, and Δ denotes the level of discrimination between actions after time μ. Note the smaller the value of σ the swifter the shift from uniform sampling to focused sampling. Also, the smaller the value of Δ the lower the level of discrimination between actions $u \in U(i)$ in state $i \in S$. We chose the above parameter values because they adequately covered the range of behaviour in our learning method when applied to the case study in the last

chapter. Not only that they were also the best choice of learning parameter sets for Method 3. Again, for $\mu = 5000$, $\sigma = 1000$ was added, resulting in two extra parameter sets (see Section 3.5.1). We made this addition because if we are restricted to such a small μ we may need to delay the switch from total sampling to focused sampling.

4.4.2 Reporting Output

The various methods that we will look at in this chapter will be evaluated according to the same framework as that used previously. Thus the tables and graphs we produce will be similar in nature to those presented in Chapter 3 (see Section 3.5).

4.4.3 Evaluating Performance

4.4.3.1 Simulations

Since in some instances we are interested in the variability of a process throughout the course of a simulation (a run), we only present results for a single seed. The results of which should not vary qualitatively from one simulation to the next. On the other hand, in other instances we are interested in variability across simulations, so we present results using different seeds. This applies to all the algorithms presented in this chapter. Single seeds are mainly used to characterise the system, whereas multiple seeds are used to observe the effect different seeds, using different learning parameter sets, have on the estimates of $v^*(\cdot)$ and $\pi^*(\cdot)$. The different results involve different analysis and output. Throughout this chapter twenty seeds are used in the multipled case labelled from 1 to 20, whereas the results taken from a single seed use the seed labelled 1.

4.4.3.2 Simulating - The Standard Version

In Section 4.6 we present results obtained from applying the methods described in Section 4.3.2 to the machine replacement problem described in Section 4.2. Section 4.6 illustrates how well the methods cope with estimating the true optimal value function $v^*(i)$ and the true optimal actions $\pi^*(i)$ defined for each state $i \in S$, the optimal solutions of which are obtained using the standard Pre-Jacobi method. As always, most of the results we present are expressed in terms of the current value function estimate and the current policy estimate. We use various criteria to identify the flaws in our *standard* algorithm when applied to a problem with this type of structure. This analysis was the main motivation behind the methodology we introduce from Section 4.7.2 onwards, which deals with more sparsely structured systems. We first analyse the results obtained by running the optimiser on its own and then analyse the results obtained when the optimiser is concurrently run with the p-learner.

4.5 Summary of Chapter

Throughout this chapter we apply various methodologies to the machine replacement problem described in Section 4.2. As in the previous chapter the two most important questions we want to address are: (1) relative to the accuracy obtained, how much slower or faster is the optimiser compared to standard DP methods ? and (2) what is the computational cost of not knowing the true state transition probabilities? In this section we give a brief summary of the important points made in the simulation sections that follow.

In Section 4.6 we compared the solutions generated by our *standard* version with that of the conventional Pre-Jacobi method. It was discovered that the Pre-Jacobi method found the true optimal solutions $v^*(i)$ and $\pi^*(i)$ in each state $i \in S$. However, our *standard* version had trouble finding good estimates to the

solutions in states 9, 10, and 11, even when the system parameters were known. After further investigation we found that the reason the optimiser obtained bad estimates in these states, was because it had difficulty sampling the higher valued states. The same was true when the system parameters were unknown. This analysis was the main motivation behind the methodology introduced in Section 4.7, which deals with more sparsely structured systems.

In Section 4.7 we describe the *indexed* optimiser. In the *indexed* optimiser each state is ranked by an index. The idea behind this method is to encourage the algorithm to visit states that need to be updated. This is based on the value of their index. A stopping criteria was also implemented. In Section 4.9 we show that the *indexed* optimiser, on average, works comparably to that of the Pre-Jacobi method when system parameters are known. Further we show that by introducing a stopping criterion in to the Pre-Jacobi method, and taking into account the fact that the optimiser is updated sequentially, the optimiser is almost as fast at finding optimal solutions. However, when we ran the *indexed* optimiser concurrently with the p-learner this was not the case.

In Sections 4.10 through to 4.12 we analysed the results using the first, second and third convergence criteria, previously motivated in Section 3.5.3.3. In Section 4.10 we show that the optimiser sampled each state fairly often but not in the p-learner. This was because we are controlling what happens in the optimiser, but the sequence of states in the p-learner is not under our control. Consequently, estimates of the true state transition probabilities were poor, and as a result the estimates of the optimal policy and optimal value function were also subject to error. In Section 4.12 we compared the root mean square error of the current value function estimate for the *indexed* algorithms with the Pre-Jacobi method. In doing so it was shown that the computational cost paid for not knowing the true state transition probabilities is fairly high. This helped motivate the development of the *coupled* version (see Section 4.13).

The *coupled* version involves the coupling of the *indexed* optimiser and p-learner. In this method, instead of observing states internally in the "black box", the *indexed* optimiser chooses the states for the "black box" and it is reset at will. We show that given that the p-learner can visit the higher valued states the algorithm's performance is highly dependent on the learning parameters used. Very few of the simulations (runs) stopped before time T. However, the current value function estimates converged to the optimal value functions in each state $i \in S$, but not without a little sampling error. Finally in Section 4.14 we investigate the performance of the *standard* method in the fully connected problem compared with that of the *indexed* method. We show that the performance of the *indexed* version is far superior when the system parameters are known, and performs at least as good when they are unknown.

4.6 The Standard Version - Convergence of the current value function and current policy estimates

In this section we present the results obtained by running the *standard* version of the optimiser both on its own and concurrently with the p-learner. We applied these algorithms to the machine replacement problem to see how well it would cope with estimating the true optimal value function $v^*(\cdot)$ and the true optimal policy $\pi^*(\cdot)$, as discussed in Section 4.4.3.2. The parameter sets $\{\mu, \sigma, \Delta\}$ are made up from the various combinations of $\mu = 5000, 10000$, and 20000, $\sigma = 100$ and 400, $\Delta = 0.003$ and 0.01, with an extra $\sigma = 1000$ added for $\mu = 5000$. More formal definitions and explanations of the learning parameters considered in this section can be found in Section 4.4.1. Note this section considers both single and multiple seeds.

4.6 The Standard Version - Convergence of the current value function and current policy estimates

Method	State $i \in S$	$v^*(i) = \widehat{v}_T(i)$	$\pi^*(i) = \widehat{\pi}_T(i)$
Pre-Jacobi	0	5.921	1
Opt Only		5.921	1
Pre-Jacobi	1	9.265	1
Opt Only		9.265	1
Pre-Jacobi	2	12.240	1
Opt Only		12.240	1
Pre-Jacobi	3	14.636	1
Opt Only		14.636	1
Pre-Jacobi	4	16.125	1
Opt Only		16.125	1
Pre-Jacobi	5	16.196	0
Opt Only		16.196	0
Pre-Jacobi	6	16.196	0
Opt Only		16.196	0
Pre-Jacobi	7	16.196	0
Opt Only		16.196	0
Pre-Jacobi	8	16.196	0
Opt Only		16.196	0
Pre-Jacobi	9	16.196	0
Opt Only		16.012	0
Pre-Jacobi	10	16.196	0
Opt Only		10.000	1
Pre-Jacobi	11	16.196	0
Opt Only		0.0000	0

Table 4.2: The table shows the current value function estimates and the current action estimates for all $i \in S$ found using the Pre-Jacobi method and when the *standard* version of the optimiser was run on its own (Opt Only) after 80000 iterations.

In Table 4.2 above we report that the Pre-Jacobi method, as usual, found the true optimal value function $v^*(i)$ and the true optimal policy $\pi^*(i)$ in all states $i \in S$. However, we can see that the optimiser on its own experienced difficulties using the parameter set $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$. This was true of all parameter sets and seeds considered in this sequence of simulations. In each case there were states for which the method did not estimate the true optimal value

function $v^*(i)$ or the true optimal policy $\pi^*(i)$. Note problems in states 9, 10 and 11. However, in spite of its recent difficulties, the optimiser when run on its own did achieve partial success. The method found the value of the threshold I in Equation (4.2.3), the cut off point between the recurrent set of states and the transient set of states. Nevertheless, it is essential to find the true optimal value function and the true optimal policy in all of the states $i \in S$ because there may come a time when we may want to begin the process (the machine) in a high valued state.

In order to further investigate the problems reported in Table 4.2, we then looked at the time interval where the current policy estimate $\widehat{\pi}_t(\cdot)$ equalled the true optimal policy $\pi^*(\cdot)$ at each time step t. We reported for all parameter sets that the current policy estimate did not equal the true optimal policy, not even for one time step. However, for other seeds we saw that the current optimal policy settled down to the true optimal policy for some parameter sets, but the current value function estimates for these runs were far from the true solution.

We then looked at the time intervals at which $\widehat{\pi}_t(i) = \pi^*(i)$ to see if some optimal actions were harder to find than others. We recall for states $i = 0$ through to 10 $\widehat{\pi}_0(i) = 1$, and for state $i = 11$ $\widehat{\pi}_0(i) = 0$. Table 4.3 below shows us that the optimiser initially experienced difficulties finding the true optimal actions in states 5 through to 9, whereas in state 10 it did not find the optimal action at all. In the previous chapter we saw that the optimiser only had difficulty choosing between a suboptimal action and the true optimal action in a particular state, if its Q-values over each action were close. In Table 4.1 above we saw that it was only in state 4 that the optimal Q-values were close. Table 4.3, on the other hand, shows us that the current action estimate in state 4 found the optimal action with ease. This was also true for states $i = 0$ to 3. We can see that the optimiser decomposed the state space in to a recurrent set of states and a transient set of states. The recurrent set of states being $\{0, 1, 2, 3, 4\}$ and the transient set of states being $\{5, 6, 7, 8, 9, 10, 11\}$. It is because of this that the

State $i \in S$	The time intervals at which $\widehat{\pi}_t(i) = \pi^*(i)$ for each individual state $i \in S$
0	0 - 80000
1	0 - 80000
2	0 - 80000
3	0 - 80000
4	0 - 80000
5	2318 - 80000
6	2287 - 80000
7	5644 - 80000
8	9029 - 80000
9	2290 - 80000
10	-
11	0 - 80000

Table 4.3: The table shows us the time intervals at which the optimiser chose each optimal action $\pi^*(i)$ in each individual state $i \in S$ using seed 1 and parameter set $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$. This result was typical of all the combinations of parameters considered.

optimiser found it difficult to sample the higher valued states. If some states are only sampled a small amount of time, regardless of whether or not we know the true values of the model, our estimates of the true optimal value function and the true optimal policy will suffer as a result. The same was true for most seeds and parameter sets considered. We will see this when we look at the behaviour of the system in Table 4.4 below.

To investigate how much influence the different parameter sets had on the method's ability to perform, we then looked at the behaviour of the system. The analysis we used was to look at the proportion of time spent sampling actions in each state and the proportion of time spent visiting each state in the optimiser, to see if some states and actions were visited less often. We report the results in Table 4.4 below. These results were typical of all seeds and parameter sets considered in this sequence of simulations. Through studying Table 4.4 below we

	$\mu = 5000\ \sigma = 1000\ \Delta = 0.01$			$\mu = 20000\ \sigma = 100\ \Delta = 0.01$		
State $i \in S$	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1
0	0.089	0.012	**0.988**	0.145	0.285	**0.715**
1	0.202	0.008	**0.992**	0.261	0.216	**0.784**
2	0.202	0.006	**0.994**	0.187	0.098	**0.902**
3	0.199	0.004	**0.996**	0.165	0.039	**0.961**
4	0.199	0.004	**0.996**	0.156	0.014	**0.986**
5	0.107	**0.998**	0.002	0.085	**0.991**	0.009
6	0.000	**0.538**	0.462	0.001	**0.400**	0.600
7	0.000	**0.500**	0.500	0.000	**0.500**	0.500
8	0 000	**0.000**	1.000	0.000	**0.417**	0.583
9	0.000	**0.333**	0.667	0.000	**0.500**	0.500
10	0.000	**0.500**	0.500	0.000	**1.000**	0.000
11	0.000	**0.000**	NA	0.000	**0.000**	NA

Table 4.4: The above table represents the percentage of time (to 3 decimal places) which we spent visiting each state and the percentage of time (to 3 decimal places) for which we spent sampling each action within each state in the optimiser. Seed 1 was used to generate the results using parameter sets { $\mu = 5000$, $\sigma = 1000$, $\Delta = 0.01$} and {$\mu = 20000$, $\sigma = 100$, $\Delta = 0.01$}. The bold figures represent the true optimal actions in each state.

can see that the optimiser found all of the true optimal actions in the recurrent state space but not in the transient state space. However, the optimiser consistently found the true optimal action in state 5. This is because after a while Method 3 helps the optimiser to choose the optimal action in each state with certainty, and the optimal action in state 5 is to replace. Therefore the state of the system very rarely ventures past state 5. This is the reason why the optimiser found it extremely difficult to estimate the true optimal value function and the true optimal actions in the higher valued states.

Since the optimiser does not perform very well even when we know all the true system parameters, we would not expect the method to work very well in conjunction with the p-learner (the method employed when the true values of

the state transition probabilities are unknown). In fact this is what we find in practice and we can see this in Table 4.5 below. We also report, as predicted in Section 4.2, when the *standard* optimiser and p-learner were run concurrently the current action estimate in state 4 oscillated between the optimal and suboptimal action until eventually settling down to the optimal action. This was true of most seeds and parameter set considered.

Method	State $i \in S$	$v^*(i) = \widehat{v}_T(i)$	$\pi^*(i) = \widehat{\pi}_T(i)$
Pre-Jacobi	0	5.921	1
Opt + PL		5.946	1
Pre-Jacobi	1	9.265	1
Opt + PL		9.258	1
Pre-Jacobi	2	12.240	1
Opt + PL		12.238	1
Pre-Jacobi	3	14.636	1
Opt + PL		14.640	1
Pre-Jacobi	4	16.125	1
Opt + PL		16.135	1
Pre-Jacobi	5	16.196	0
Opt + PL		16.213	0
Pre-Jacobi	6	16.196	0
Opt + PL		16.294	0
Pre-Jacobi	7	16.196	0
Opt + PL		16.595	0
Pre-Jacobi	8	16.196	0
Opt + PL		16.324	0
Pre-Jacobi	9	16.196	0
Opt + PL		16.156	0
Pre-Jacobi	10	16.196	0
Opt + PL		0.000	1
Pre-Jacobi	11	16.196	0
Opt + PL		0.000	0

Table 4.5: The table shows the current value function estimates and the current action estimates for all $i \in S$ found using the Pre-Jacobi method and when the *standard* version of the optimiser and p-learner were run concurrently (Opt + PL) for 80000 iterations.

4.7 The Indexed Optimiser

4.7.1 Introduction

This second part of the chapter discusses the *indexed version* of the optimiser. In this section we will discuss a new stochastic calculation technique used to sample states in the optimiser, and the motivation behind it. We use this routine in the sequence of simulations that follow. Similar ideas are discussed in Kaelbling *et al.* (1996) but without the focused motivation of our empirical study (see Section 1.7.5.1).

In the last section we saw that the methodology that worked for the fully connected case, choosing both states and actions in the optimiser, does not necessarily work for systems which are sparsely connected. The root problem being the *standard* optimiser made too few visits to the higher valued states, even when all the system parameters were known. Thus not all approximations of the true optimal value function $v^*(i)$ and the true optimal policy $\pi^*(i)$ for each state $i \in S$ were good.

The theory in Chapter 5 tells us that if each state is sampled often enough and if all the system parameters are known, the current value function estimate and the current policy estimate will eventually converge to the true optimal value function and the true optimal policy respectively. To ensure this, we could sample each state uniformly at random. However, in doing so we would be in danger of wasting far too much time visiting redundant states until the end of the simulation horizon T. Redundant states are states at which the current value function estimate has already converged to the true optimal value function estimate. Our *indexed* version, on the other hand, updates states where the computation is needed most by means of a built-in stopping criteria.

In Sections 4.7.2 through to 4.7.4 we will discuss methodology developed to overcome problems evident in the previous section. The methodology we propose

is known as the *indexed optimiser*, which incorporates a routine called the Index Method, which we discuss below. In Section 4.7.2 we discuss the quantities used in this method, known as indices, which are defined for each state $i \in S$, and we describe how they are updated at each time t. In Section 4.7.3 we show, given that the values of these quantities are known, that we can use the information to sample states. Section 4.7.4 discusses a dynamic stopping criterion defined in terms of the indices. We describe the procedure of the methodology at each iteration in Section 4.8.1, and in Section 4.8.4.2 we evaluate their performance. In Sections 4.9 through to 4.12 we present results when system parameters are both known and unknown. Finally we end the second part of this chapter comparing the performance of the *indexed* method with that of the *standard* method in the fully connected case.

4.7.2 The Index Method

In this section we describe a routine that helps the optimiser decide which state to update next, but without choosing actions. The method is written in terms of the optimiser when run on its own at time t. If used when the optimiser is run concurrently with the p-learner, the current value function estimate $\widehat{v}_{t+1}(\cdot)$, the current policy estimate $\widehat{\pi}_{t+1}(\cdot)$ and the true state transition probabilities $p_{ij}(u)$ should be substituted for $\widetilde{v}_{t+1}(\cdot)$, $\widetilde{\pi}_{t+1}(\cdot)$ and $\hat{p}^t_{ij}(u)$ respectively.

In this method instead of sampling each state uniformly at random, we propose to sample each state according to the distribution over what we call indices τ_i, assigned to each state $i \in S$. Each index is a numerical value which changes over time, and is dependent on the changes of the current value function estimate over time. As t increases, the current value function estimates get closer and closer, and their corresponding indices get smaller and smaller. Ultimately the indices become so small that the algorithm decides to terminate computing.

The index method was designed as a cheap heuristic that accurately measures

how often states have been visited. The higher the value of the index τ_i, the more out-of-date the current value function at state i becomes. They are used to determine the probability that the current value function estimate at state $i \in S$ is to be updated, thus the optimiser is encouraged to visit a high valued index over a low valued index (see Section 4.7.3).

One of the features of this index routine is that if updating the current value function estimate at state $i \in S$ does not have much effect, from one time step to the next, then we are not particularly encouraged to visit state i again. If this is true for all states then it is a good enough reason to stop computing. We discuss this in Section 4.7.4. The aim of using the index routine is to maximise the values of our efforts and not to neglect states for too long, unless its corresponding current value function estimates have already converged.

To define an index τ_i for all $i \in S$ we use information available to us that is quick and easy to compute. Let us assume we are at time t, and we have information about current value function estimate $\widehat{v}_t(\cdot)$, the set of true state transition probabilities $p_{ij}(u)$ for all $i, j, \in S$ and $u \in U(i)$, and the current state $i_t = i$. Having computed the current value function estimate $\widehat{v}_{t+1}(\cdot)$ and the current policy estimate $\widehat{\pi}_{t+1}(\cdot)$ using Equations (2.2.5) and (2.2.6) respectively, we then need to consider updating the indices ready to sample the next state at time $t+1$.

Consider updating the current value function estimate for state $j \in S$ at time $t+1$ where,

$$\widehat{v}_{t+2}(j) = \min_{u \in U(j)} \left\{ c_j(u) + \alpha \sum_l p_{jl}(u) \widehat{v}_{t+1}(l) \right\} \quad \text{and,} \tag{4.7.7}$$

$$\widehat{v}_{t+1}(j) = \min_{u \in U(j)} \left\{ c_j(u) + \alpha \sum_l p_{jl}(u) \widehat{v}_t(l) \right\}. \tag{4.7.8}$$

Subtracting Equation (4.7.8) from Equation (4.7.7) we get,

$$\begin{aligned}\widehat{v}_{t+2}(j) - \widehat{v}_{t+1}(j) &= \alpha \sum_{l} p_{jl}(u) \{\widehat{v}_{t+1}(l) - \widehat{v}_t(l)\}, \\ &= \alpha p_{ji}(u) \{\widehat{v}_{t+1}(i) - \widehat{v}_t(i)\}, \qquad (4.7.9)\end{aligned}$$

since,

$$\widehat{v}_{t+1}(l) = \widehat{v}_t(l) \quad \forall\, l \neq i \in S,$$

as shown using Equations (2.2.4) and (2.2.5) in Section 2.2. Equation (4.7.9) is therefore an estimate of how much change there might have been if we had visited a state $j \in S$ instead of the current state i. We use this information to update the indices. Note in Equations (4.7.7) and (4.7.8) we have assumed the action u that minimises both equations is the same, as the method is only meant to be a quick and easy heuristic. Taking this on board, we can define indices τ_j for all states $j \in S$ and time $t = 0, 1, 2, \ldots$.

Initially, we can set all the indices τ_j for all states $j \in S$ at stage 0 to a large finite value N so that each state is equally likely to be chosen (see Section 4.7.3 below). Then for each state $j \in S$ at stage $k = 0, 1, 2, \ldots$, a sensible index τ_j after we have computed $\widehat{v}_{t+1}(i)$ can be defined as follows,

$$\begin{aligned}\tau_j &= 0 \qquad \text{for } j = i, \\ &= \tau_j + \sigma_{ji}|\widehat{v}_{t+1}(i) - \widehat{v}_t(i)| \quad \forall\ j \neq i, \qquad (4.7.10)\end{aligned}$$

where,

$$\sigma_{ji} = \max_{u \in U(j)} \alpha p_{ji}(u). \qquad (4.7.11)$$

After we update the indices we check to see if the current value function estimates are worth updating for one more iteration using Equation(4.7.15) below. If Equation (4.7.15) is satisfied then computing terminates, otherwise the indices

computed above are used to choose the next current state at time $t+1$ using Equation (4.7.14) below.

Note that if the optimiser and p-learner are run concurrently then Equations (4.7.10) and (4.7.11) must be changed to,

$$\begin{aligned} \tau_j &= 0 \qquad \text{for } j = i, \\ &= \tau_j + \hat{\sigma}_{ji}|\widetilde{v}_{t+1}(i) - \widetilde{v}_t(i)| \quad \forall\ j \neq i, \end{aligned} \tag{4.7.12}$$

where,

$$\hat{\sigma}_{ji} = \max_{u \in U(j)} \alpha \hat{p}_{ji}(u). \tag{4.7.13}$$

Making the indices τ_j, $j \neq i$, an additive function until the optimiser actually uses them, means that a state that has not been visited for a while is going to be dominant eventually. Thus, the higher the value of τ_j the more likely the current value function estimate at state j is to be updated. Therefore once we visit a state, its index is set to zero and it is out of the competition for a while at least, because its corresponding current value function estimate has just been updated. If τ_j is large, this tells the optimiser, either because of recent changes in the states it is connected to or because it has not been looked at for a while, that the current value function estimate needs updating.

If there is a difference between $\widehat{v}_t(i)$ and $\widehat{v}_{t+1}(i)$, the parameter defined above in Equation (4.7.11) is the upper bound on a change induced by $\widehat{v}_{t+2}(j)$. We use the upper bound in Equation (4.7.10) because if τ_j is small for all $i, j \in S$ the current value function estimate is not going to change very much at each stage of the computation, and it will settle down to the true optimal value function estimate eventually.

. In some respects, the Gauss Seidel method is similar to our algorithm in that it calculates the current value function estimates with its most recent information. Essentially, this is what the indices are doing. Each time we update the current

value function estimate at a particular state i we update the indices, and this effects our updating of the current value function estimate for each other state $j \neq i \in S$. Thus, we are updating current value function estimate at stage k and creating a quick index of what the maximum change might mean for other states $j \neq i$ defined by σ_{ji} : a change in state i for a per unit change in j.

4.7.3 Sampling states

In this section we describe a sampling routine that the optimiser uses to update the current value function estimate.

Since the indices τ_i for each state $i \in S$ measure the degree of neglect individual states have had, we use this information to sample states. In our method we update the current value function estimate at state $i \in S$ according to the distribution,

$$\min\left\{\tau_i\left\{\sum_{l\in S}\tau_l\right\}^{-1},1\right\}. \tag{4.7.14}$$

Thus if the value of an index for a particular state is relatively large compared to the other states, it is probable that its corresponding state will be visited shortly.

4.7.4 Stopping Criteria

In this section we describe the stopping criterion used in the index method as discussed in Section 4.7.1.

We may want to terminate a simulation before the end of the fixed horizon time T, if all the changes in the current value function estimate for all the states $i \in S$ become too small. At the start of each simulation all the indices are set to a large value N, but as time passes the current value function estimates converge to a point and consequently the indices settle down and tend to zero. However, if for a particular update on a particular occasion the change in the corresponding

current value function estimate is small then we should not necessarily stop, since the effect of the previous updates may not have propagated through to the other states. On the other hand, if all the anticipated changes in the value functions are small, then this is a plausible argument to stop.

We introduce a tolerance parameter, a threshold denoted by D, which is a constant value, and we propose to stop when,

$$\sum_{j \in S} \tau_j < D. \tag{4.7.15}$$

Obviously, the more accurate we want our approximations to be, the smaller we set the value of the threshold D.

4.8 The Methodology Used In The Second Part Of This Chapter

4.8.1 Review

In this section we briefly discuss the *indexed* version of the methodology used in the second part of this chapter. For a brief review of the how the optimiser and p-learner generally work in conjunction with one another, please refer back to Section 4.3.1. For a more detailed account of how the methods work using the index method at each iteration, see Sections 4.8.2 and 4.8.3 below. We first describe what happens at each iteration when the optimiser is run on its own and then when the optimiser is run concurrently with the p-learner.

4.8.2 Running the optimiser on its own

Consider first the case when the immediate costs $c_i(u)$ and the true state transition probabilities $p_{ij}(u)$ are assumed known for each state $i, j \in S$ and action $u \in U(i)$, in which case we proceed by running the optimiser on its own.

4.8 The Methodology Used In The Second Part Of This Chapter

As in all the algorithms discussed in this thesis the choice of the initial values is somewhat arbitrary. In practice at time $t = 0$, we set the initial value function estimates $\widehat{v}_0(l) = 0$ for each state $l \in S$; we set the action estimates $\widehat{\pi}_0(l) = 1$ for states $l = 0$ through to 10 and $\widehat{\pi}_0(11) = 0$; each index τ_l corresponding to each state $l \in S$ at time $t = 0$ is set to some large value N; we set the stopping threshold D to 1×10^{-5} and at each run we choose an initial state i_0 at random from the probability distribution of τ_l for all $l \in S$.

At each time t we assume we have information about the current state $i_t = i$, the immediate costs $c_i(u)$ and the set of true state transition probabilities $p_{ij}(u)$ for all states $i, j \in S$ and actions $u \in U(i)$. We also assume we know the current value function estimate $\widehat{v}_t(l, u)$ evaluated at each state $l \in S$ and each action $u \in U(l)$, the current value function estimate $\widehat{v}_t(\cdot)$, the value of the stopping threshold D and the value of τ_l for each $l \in S$.

At each iteration t, the optimiser uses the Asynchronous DP algorithm defined in Equation (2.2.4) to update $\widehat{v}_{t+1}(l, u)$ at each state $l \in S$ and possible actions $u \in U(l)$. The estimate $\widehat{v}_{t+1}(i, u)$ at time t is calculated using the true state transition probabilities $p_{ij}(u)$, the immediate costs $c_i(u)$, and the current value function estimate $\widehat{v}_t(\cdot)$, while the other value function estimates are updated by setting $\widehat{v}_{t+1}(i', u) = \widehat{v}_t(i', u)$ at states $i' \neq i$. Immediately after updating $\widehat{v}_{t+1}(l, u)$ the optimiser synchronously updates the current value function estimate $\widehat{v}_{t+1}(\cdot)$ and the current policy estimate $\widehat{\pi}_{t+1}(\cdot)$ by setting $\widehat{v}_{t+1}(l) = \min_{u \in U(i)}\{\widehat{v}_{t+1}(l, u)\}$ and $\widehat{\pi}_{t+1}(l) = \arg\min_{u \in U(l)} \{\widehat{v}_{t+1}(l, u)\}$ for each state $l \in S$, as defined in Equations (2.2.5) and (2.2.6) respectively.

Having updated the current value function estimate and the current policy estimate the optimiser then proceeds to update the values of the indices, τ_l, for each state $l \in S$ using Equation (4.7.10). After the indices have been updated the optimiser checks to see whether it is worth updating the current value function estimate and the current policy estimate for one more iteration using

Equation (4.7.15). If the condition in Equation (4.7.15) is satisfied the optimiser will stop computing, otherwise it will generate the next state i_{t+1} using Equation (4.7.14) below ready for the next iteration.

4.8.3 Running the optimiser concurrently with the p-learner

Next consider the case when the immediate costs $c_i(u)$ are assumed known but the true state transition probabilities $p_{ij}(u)$ are assumed unknown for each state $i, j \in S$ and action $u \in U(i)$, in which case we proceed by running the optimiser concurrently with the p-learner. Please note that when the optimiser and p-learner are run concurrently, the sequence of states generated by the p-learner may be different to the sequence of states generated by the optimiser.

As in the known P case the starting values are chosen arbitrarily. In practice, at time $t = 0$, in the optimiser, we set the initial value function estimates $\widetilde{v}_0(l) = 0$ for each state $l \in S$; we set the action estimates $\widetilde{\pi}_0(l) = 1$ for states $l \in S$ listed between 0 to 10 (inclusive) and $\widetilde{\pi}_0(11) = 0$; we set the state transition probability estimates $\hat{p}^0_{lm}(u) = 1/2$ for each state $l \in S$ given state $m \in S$ is an immediate possible successor, otherwise $\hat{p}^0_{lm}(u) = 0$; we set each index τ_l corresponding to each state $l \in S$ to some large value N; we set the stopping threshold to 1×10^{-5} and at each run we choose an initial state i_0 at random from the probability distribution of τ_l for all $l \in S$.

In the optimiser at each time t we assume we have information about the current state $i_t = i$, the immediate costs $c_i(u)$ and the set of current state transition probability estimates $\hat{p}^t_{ij}(u)$ for all states $i, j \in S$ and actions $u \in U(i)$, the current value function estimate $\widetilde{v}_t(l, u)$ evaluated at each state $l \in S$ and each action $u \in U(l)$, the current value function estimate $\widetilde{v}_t(\cdot)$ and the value of each index τ_l for each state $l \in S$.

At each iteration t, the Asynchronous DP algorithm defined in Equation (2.4.16)

is used to update $\widetilde{v}_{t+1}(l,u)$ at all states $l \in S$ and possible actions $u \in U(l)$. The optimiser calculates $\widetilde{v}_{t+1}(i,u)$ using the current state transition probability estimates $\hat{p}_{ij}^{t}(u)$, the immediate costs $c_i(u)$, and the current value function estimates $\widetilde{v}_t(\cdot)$, while the other value function estimates are updated by setting $\widetilde{v}_{t+1}(i',u) = \widetilde{v}_t(i',u)$ at states $i' \neq i$. Having calculated $\widetilde{v}_{t+1}(l,u)$ the optimiser then proceeds to simultaneously update the current value function estimate $\widetilde{v}_{t+1}(\cdot)$ and the current policy estimate $\widetilde{\pi}_{t+1}(\cdot)$ using Equations (2.4.17) and (2.4.18) respectively, by setting $\widetilde{v}_{t+1}(l) = \min_{u \in U(l)} \{\widetilde{v}_{t+1}(l,u)\}$ and $\widetilde{\pi}_{t+1}(l) = \arg\min_{u \in U(l)} \{\widetilde{v}_{t+1}(l,u)\}$ for each state $l \in S$.

Having calculated the current value function estimate and the current policy estimate at time t the optimiser uses Equation (4.7.12) to update each index τ_l for each state $l \in S$. Subsequently the optimiser monitors the quality of its current estimates by looking to see if the condition in Equation (4.7.15) is satisfied. If Equation (4.7.15) is satisfied computing in the optimiser discontinues, otherwise the next state i_{t+1} is generated for the next iteration. Even if computing in the optimiser has stopped the p-learner is iterated for one last time, but only for the sake of presenting results in the remainder of this chapter. This is only because in some diagrams we want to present results comparing the quality of the current value function with the quality of the current quality of the current model, at each iteration.

After the optimiser has been iterated the p-learner then updates the current state transition probabilities defined in Equation (4.3.5) using the same procedures as the ones described in the *standard* version (see Section 4.3.2 above).

4.8.4 Evaluating Performance

4.8.4.1 Simulations

For an explanation as to why we observe results for both single and multiple seeds in the following sequence of simulations, please refer back to Section 4.4.3.1. Note that in the remaining chapter, H_N denotes the number of the decision epoch at which a particular run ends using seed N.

4.8.4.2 Simulating - The Indexed Version

In Section 4.9 we show that the performance of the *indexed* optimiser when run on its own is a great improvement on the *standard* optimiser when applied to a "sparsely connected" system. We compare the method's estimates with those gained by the Pre-Jacobi method. As well as showing how close our estimates are to their corresponding optimal solutions, we introduce a stopping criterion in the Pre-Jacobi method to compare the speed and accuracy of its estimates with ours. Having shown that the *indexed* optimiser works effectively on its own, it is worthwhile investigating how it works in conjunction with the p-learner.

Since in Chapter 3 we saw that the performance of the p-learner depended to some extent on the learning parameter sets $\{\mu, \sigma, \Delta\}$ used, in Sections 4.9 through to 4.12 we will now look at the performance of the *indexed* optimiser when run concurrently with the p-learner over a range of parameter sets.

In Sections 4.10 and 4.11 we characterise the behaviour of the system using the first convergence criterion, previously discussed in Section 3.5.3.3. In Section 4.10 this is done by analysing the proportion of time spent visiting and observing states both in the optimiser and the p-learner, and the proportion of time spent sampling actions within each state in the p-learner. Then in Section 4.11 having characterised the behaviour of the system using different learning parameters, we look graphically at whether the current policy estimates settle down to an optimal policy or a suboptimal policy. Illustrated in these graphs are the time points at

which the computation terminates. In Section 4.12 we conclude the analysis by looking at the second and third convergence criterion, again previously motivated in Section 3.5.3.3. Included in this, is a comparison of the root mean square error of the current value function estimate for three different methods; the *indexed* optimiser on its own, the *indexed* optimiser run concurrently with the p-learner, and the conventional Pre-Jacobi method.

4.9 Running the indexed optimiser on its own - Convergence of the current value function and the current policy

Method	State $i \in S$	$v^*(i) = \widehat{v}_{H_1}(i)$	$\pi^*(i) = \widehat{\pi}_{H_1}(i)$
Pre-Jacobi	0	5.921	1
Opt Only		5.921	1
Pre-Jacobi	1	9.265	1
Opt Only		9.265	1
Pre-Jacobi	2	12.240	1
Opt Only		12.240	1
Pre-Jacobi	3	14.636	1
Opt Only		14.636	1
Pre-Jacobi	4	16.125	1
Opt Only		16.125	1
Pre-Jacobi	5	16.196	0
Opt Only		16.196	0
Pre-Jacobi	6	16.196	0
Opt Only		16.196	0
Pre-Jacobi	7	16.196	0
Opt Only		16.196	0
Pre-Jacobi	8	16.196	0
Opt Only		16.196	0
Pre-Jacobi	9	16.196	0
Opt Only		16.196	0
Pre-Jacobi	10	16.196	0
Opt Only		16.196	0
Pre-Jacobi	11	16.196	0
Opt Only		16.196	0

Table 4.6: The table shows the current value function estimates and the current action estimates for all $i \in S$ found using the Pre-Jacobi method and when the *indexed* optimiser was run on its own (Opt Only). A fixed finite horizon time $T = 80000$ was used in both methods but the symbol H_N denotes the number of the decision epoch at which a run ends for seed N.

In this section we present the results obtained when running the *indexed*

optimiser on its own to see how well it would cope with estimating the true optimal value function $v^*(\cdot)$ and the true optimal policy $\pi^*(\cdot)$, as discussed in Section 4.8.4.2. In this section both single and multiple seeds are considered.

Table 4.2 shows that, unlike the previous section, the *indexed* optimiser was run on its own produced the same results as those produced when using the Pre-Jacobi method. In addition to this when the *indexed* optimiser was run on its own the current policy estimate converged to the true optimal policy after 51 iterations and the current value function estimate converged to the true optimal value function after 461 iterations, when it stopped. This is apparent when we look at the time intervals where the current policy estimate $\widehat{v}_t(\cdot)$ equalled the true optimal policy $v^*(\cdot)$ at each time t, as shown in Table 4.7 below.

Method	The time intervals at which $\widehat{\pi}_t(i) = \pi^*(i)$ for all $i \in S$
Opt Only	26 - 31 51 - 461

Table 4.7: The table shows us the time intervals at which the indexed optimiser when run on its own (Opt Only) chose the true optimal actions in all states $i \in S$. Similar results were obtained for runs using other seeds.

We looked at the average of $\hat{v}_{H_N}(\cdot)$ and $\hat{\pi}_{H_N}(\cdot)$ over the 20 different seeds to examine the consistency and robustness of our method. We report similar results to the ones quoted above. On average the true optimal value function and the true optimal policy were found after 441 and 84 iterations respectively. This shows that the *indexed* optimiser when applied to the machine replacement problem is equivalent to the *standard* optimiser when applied to the "fully connected" case, that is, optimal solutions were found every time.

To compare the above two methods performances, we introduce a simple stopping criterion for the Pre-Jacobi method which is similar to the one used in the *indexed* optimiser, described in Section 4.7.4. In both methods the stopping

criteria is set up so that computing stops once the current value function estimate equals the true optimal value function in each state $i \in S$, to three decimal places. The stopping criterion in the Pre-Jacobi method is defined in Equation (4.9.16) below. It stops computing when,

$$|v_{t+1}(\cdot) - v_t(\cdot)| < 1 \times 10^{-5}. \tag{4.9.16}$$

In fact the Pre-Jacobi method found the the true optimal policy after 7 iterations and the true optimal value function after 46 iterations, when it stopped. Therefore the *indexed* optimiser when run on its own achieved favourable results. This is good considering the method only updates the current value function estimate and the current policy estimate at one state per time step, and the Pre-Jacobi method updates the current value function estimate and the current policy estimate at twelve states per time step. Note $46 \times 12 = 552$ and $7 \times 12 = 84$. We recall that, on average, the indexed optimiser when run on its own found the true optimal value function and the true optimal policy after 441 and 84 iterations respectively. It must be noted, however, that we are not exactly comparing like with like as the stopping criteria in Equation (4.9.16) is defined slightly differently to Equation (4.7.11), but comparing them in this way does give us a good idea of their ability to perform.

In order to investigate the results gained using the indexed optimiser even further, we then looked at the individual time steps where $\widehat{\pi}_{H_1}(i) = \pi^*(i)$ for each state $i \in S$. These results tell us which states, if any, had trouble finding their corresponding true optimal actions. We report the results presented in Table 4.8 below using seed 1 (though the results are similar to those for the other seeds). Table 4.8 below tells us that the optimiser found all of the true optimal actions in each individual state with ease especially in states $\{0, 1, 2, 3, 11\}$. As predicted in Section 4.2, state 4 initially oscillated between choosing the suboptimal action and the true optimal action, but the current action estimate eventually settled down to the true optimal action with no major problems. This was to be expected

after looking at the optimal Q-values for this problem, as reported in Table 4.1 above.

State $i \in S$	The time intervals at which $\widehat{\pi}_t(i) = \pi^*(i)$ for each individual state $i \in S$
0	0 - 461
1	0 - 461
2	0 - 461
3	0 - 461
4	0 - 31 51 - 461
5	26 - 461
6	19 - 461
7	17 - 461
8	7 - 461
9	6 - 461
10	13 - 461
11	0 - 461

Table 4.8: The table shows us the time intervals at which the indexed optimiser chose each optimal action $\pi^*(i)$ in each individual state $i \in S$ before computing terminated, using seed 1. Similar results were obtained when other seeds were used.

In the last section we found out that the reason that the *standard* optimiser had difficulty obtaining good estimates of the true optimal value function $v^*(i)$ and the true optimal policy $\pi^*(i)$ in all of the states $i \in S$, was that the optimiser did not sample the higher valued states very often. We now characterise the behaviour of the system by looking at the proportion of time the indexed optimiser spent visiting individual states $i \in S$, to see if the same was experienced in this method. Table 4.9 shows that this was not the case. This is why the true optimal value function estimates $v^*(i)$ and the true optimal action estimates $\pi^*(i)$ were found in all states. We report that this was true for each seed considered. Note all of the states were visited fairly uniformly except in state 0, where the current

value function estimate converged fairly quickly, because its optimal Q-value was small (see Table 4.1).

State $i \in S$	Proportion of time in state i
0	0.054
1	0.085
2	0.102
3	0.087
4	0.080
5	0.085
6	0.082
7	0.089
8	0.085
9	0.080
10	0.091
11	0.080

Table 4.9: The above table represents the percentage of time (to 3 decimal places) for which we spent visiting each state when the indexed optimiser was run on its own. Seed 1 was used to generate the results. Similar results were obtained for other seeds.

Now we have an optimiser that works effectively when applied to the machine replacement problem, in the next section we investigate how it works in conjunction with the p-learner.

4.10 Running the indexed optimiser concurrently with the p-learner - Sampling States and Actions

In the following two sections we look at the first of the convergence criteria, the convergence of the current policy estimate when the *indexed* optimiser is run concurrently with the p-learner, as discussed in Section 4.8.4.2. In this first section we characterise the behaviour of the system in terms of the proportion of time spent visiting and observing states in the *indexed* optimiser and p-learner respectively, and the proportion of time spent sampling actions within each state in the p-learner. The parameters sets considered for $\{\mu, \sigma, \Delta\}$ are the ones obtained through looking at the various combinations of parameters for $\mu = 5000, 10000,$ and 20000, $\sigma = 100$ and 400, $\Delta = 0.003$ and 0.01 with an extra $\sigma = 1000$ included for $\mu = 5000$ (see Section 4.4.1). We recall that μ denotes the amount of learning we are willing to do, σ denotes how quickly we jump from total sampling to focused sampling, and Δ determines the level of discrimination between actions $u \in U(i)$ in each state $i \in S$. Note the bigger the value of σ the slower we move from total sampling to focused sampling, and the larger the value of Δ the larger the level of discrimination between actions in each state. In this section only single runs are considered.

Tables 4.10 and 4.11 below illustrate (for parameter sets $\{\mu = 5000, \sigma = 1000, \Delta = 0.01\}$ and $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$) the proportion of time for which we visited and observed each state in the *indexed* optimiser and the p-learner respectively, and in addition to this, Table 4.11 also illustrates the proportion of time for which we sampled each action in each particular state for the two different parameter sets. These two tabulated sets of results exemplify the typical differences in the behaviour of sampling states in both the *indexed* optimiser and p-learner, irrespective of the parameter sets used in the p-learner.

4.10 Running the indexed optimiser concurrently with the p-learner - Sampling States and Actions

Table 4.11 further illustrates the influence certain parameters have on the system's ability to sample actions in each particular state. Note all of the bold figures in this table correspond to the true optimal actions $\pi^*(i)$ in each state $i \in S$. Also, the proportion of time in each state was taken to three decimal places. Therefore in the states where the p-learner observed a state a small number of times relative to the length of time before computing stopped, the proportion of times spent is recorded as 0.000. This can be seen in the results reported in Table 4.11 for state 6, using parameter set $\{ \mu = 5000, \sigma = 1000, \Delta = 0.01\}$. The proportion of time spent observing state 6 is 0.000 but the proportion of time spent sampling action 0 in state 6 is 1.000.

In Tables 4.10 and 4.11 we can see that the only difference between the number of visits across each state in the *indexed* optimiser and the p-learner, is that in the *indexed* optimiser the proportion of visits to each state is more evenly spread. In the *indexed* optimiser we can see that all the visits to each state are fairly uniform except in state 0. This was typical of all parameter sets and seeds used in this sequence of simulations. This is not too surprising since the optimal Q-value in state 0 is fairly small compared to the other states (see Table 4.1). Thus given all initial values function estimates $\widetilde{v}_0(i)$ were initially set to zero for each state $i \in S$, it would make sense that the current value function estimate in state 0 would be one of the first functions to converge to its corresponding true value; if not the first. Also, the index method in the *indexed* optimiser was set up so that if a current value function estimate had already settled down to its corresponding optimal value, it would be highly unlikely that we would ever visit that particular state again.

Table 4.11, on the other hand, shows that the p-learner had similar problems to the *standard* optimiser, reported in Section 4.6. It had difficultly observing the higher valued states, again because the current policy estimate decomposed the state space in to a recurrent state space and a transient one. These difficulties in the p-learner were present irrespective of the learning parameters and seeds used.

	$\mu = 5000\ \sigma = 1000\ \Delta = 0.01$	$\mu = 20000\ \sigma = 100\ \Delta = 0.01$
State $i \in S$	Proportion of time in state i	Proportion of time in state i
0	0.059	0.061
1	0.082	0.087
2	0.095	0.100
3	0.094	0.090
4	0.098	0.087
5	0.086	0.087
6	0.084	0.082
7	0.083	0.085
8	0.084	0.081
9	0.083	0.083
10	0.083	0.085
11	0.070	0.070

Table 4.10: The above table represents the percentage of time (to 3 decimal places) which we spent visiting each state in the *indexed* optimiser (when concurrently run with the p-learner). Seed 1 was used to generate the results using parameter sets $\{\ \mu = 5000,\ \sigma = 1000,\ \Delta = 0.01\}$ and $\{\mu = 20000,\ \sigma = 100,\ \Delta = 0.01\}$.

This is because although we are controlling what happens in the optimiser, the sequence of states visited in the p-learner is not under our control.

To observe the effects different parameters had on sampling states in the system we report the results quoted in Table 4.11. Note that the results in Table 4.11 need to be interpreted with care. In the last chapter we showed that for a small value of μ we would expect to see the p-learner focus on what the optimiser perceived as being the true optimal policy for the majority of the time. For the parameter set $\{\mu = 5000, \sigma = 1000, \Delta = 0.01\}$ we can see that in the states that the p-learner visited fairly often, that is states 0 through to 5, this was true. The p-learner spent a great deal of time visiting the true optimal action. For the parameter set $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$ we expected to see the p-learner sample the current action estimates in states 0 through to 5

4.10 Running the indexed optimiser concurrently with the p-learner - Sampling States and Actions

	$\mu = 5000\ \sigma = 1000\ \Delta = 0.01$			$\mu = 20000\ \sigma = 100\ \Delta = 0.01$		
State $i \in S$	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1	Proportion of time in state i	Proportion of time sampling $u \in U(i)$ 0	1
0	0.080	0.017	**0.983**	0.237	0.518	**0.482**
1	0.210	0.017	**0.983**	0.484	0.504	**0.496**
2	0.203	0.012	**0.988**	0.180	0.519	**0.481**
3	0.197	0.008	**0.992**	0.063	0.506	**0.494**
4	0.196	0.010	**0.990**	0.024	0.475	**0.525**
5	0.115	**0.997**	0.003	0.008	**0.462**	0.538
6	0.000	**1.000**	0.000	0.003	**0.592**	0.408
7	0.000	**0.000**	0.000	0.001	**0.727**	0.273
8	0.000	**0.000**	0.000	0.000	**0.500**	0.500
9	0.000	**0.000**	0.000	0.000	**1.000**	0.000
10	0.000	**0.000**	0.000	0.000	**0.000**	0.000
11	0.000	**0.000**	NA	0.000	**0.000**	NA

Table 4.11: The above table represents the percentage of time (to 3 decimal places) for which we spent visiting each state and the percentage of time (to 3 decimal places) for which we spent sampling each action within each state in the p-learner (when concurrently run with the *indexed* optimiser). Seed 1 was used to generate the results using parameter sets $\{\ \mu = 5000,\ \sigma = 1000,\ \Delta = 0.01\}$ and $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$. The bold figures represent the true optimal actions in each state.

approximately 80% of the time, as in the previous chapter. Table 4.11 shows that this was not the case. The p-learner sampled the actions in these states with approximately equal probability. Nevertheless, when we looked at this more closely computing stopped after 46356 iterations for the parameter set $\{\mu = 5000, \sigma = 1000, \Delta = 0.01\}$. Thus the p-learner had an ample amount of time in which to focus on the current action estimates in the recurrent class of states because the specified learning time was 5000 iterations. On the other hand, for the parameter set $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$ computing stopped after 14423 iterations. Therefore the p-learner using $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$ did not have enough time to focus on the current action estimates in the recurrent class of states, as the method stopped during the learning period.

We conclude that in states 0 through to 5 the parameter sets would have sampled the current action estimates for approximately the same amount of time as in the previous chapter, except the computing stopped before the fixed simulation horizon time was reached. This was true for all parameter sets and seeds considered. This section has shown that the p-learner does not sample the current action estimates in the higher valued states because of the different learning parameter sets used, but because of an underlying deficiency in the p-learner when faced with sparsely structured systems. In the next section we will look at how this deficiency affected the optimiser's ability to focus on the optimal policy.

4.11 Running the indexed optimiser concurrently with the p-learner - Convergence of the current policy estimate

This section is the last of the two sections concerned with the first of the convergence criteria, the convergence of the current policy estimates, as discussed in Section 4.8.4.2. In the last section we established that the p-learner is still experiencing difficulties, in that it is not updating the higher valued states. In this section we look to see if these difficulties have propagated through to the convergence of the current policy estimate. These results are compared with the optimal solutions gained from when the *indexed* optimiser was run on its own, discussed in Section 4.9. The learning parameter sets considered in this section consist of the various combinations of parameters for $\{\mu, \sigma, \Delta\}$ taken from $\mu = 5000, 10000$ and 20000, $\sigma = 100$ and 400, $\Delta = 0.003$ and 0.01, with an extra $\sigma = 1000$ included for $\mu = 5000$ (see Section 4.4.1). We recall that μ controls the amount of learning, σ controls the transition from total sampling to focused sampling, and Δ controls the level of discrimination between states after time μ. In this section we consider results from both single and multiple seeds depending

on the required output (see Section 4.4.2).

Tables 4.12 and 4.13 below illustrate the typical problems confronted by the *indexed* optimiser when run concurrently with the p-learner. The tabulated results in Table 4.12 consist of the time intervals where the current action estimates converged to the true optimal actions in each individual state using different learning parameter sets corresponding to $\mu = 20000$. Table 4.13 reports, not only the

State $i \in S$	The time intervals at which $\widehat{\pi}_t(i) = \pi^*(i)$ for each individual state $i \in S$
0	0 - 14423
1	0 - 14423
2	0 - 14423
3	0 - 14423
4	0 - 32 54 - 61 1578 - 1592 1610 - 3857 3938 - 3941 4132 - 4229
5	28 - 14423
6	23 - 14423
7	16 - 14423
8	12 - 14423
9	3 - 14423
10	14 - 14423
11	0 - 14423

Table 4.12: The table shows us the time intervals at which the *indexed* optimiser when run concurrently with the p-learner chose each optimal action $\pi^*(i)$ in each individual state $i \in S$ before computing terminated, using seed 1. Similar results were obtained when other seeds were used.

last time intervals where the optimiser's current policy estimates settled down to the true optimal policy using the parameter set $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$ for *multiple seeds, but also* quotes the time step where the computing stopped in each case. Figures 4.1 and 4.2, on the other hand, display *the time intervals*

where the current policy estimate equalled the true optimal policy for different learning parameters. These graphs were first introduced in the previous chapter (see Section 3.9).

Seed N	The last time intervals at which $\widetilde{\pi}_{H_N}(\cdot) = \pi^*(\cdot)$ for different seeds
1	4132 - 4229
2	12689 - 62532
3	16288 - 66356
4	20470 - 34510
5	222 - 30376
6	26863 - 28671
7	43194 - 43341
8	10499 - 42067
9	26199 - 45608
10	106 - 35478
11	10336 - 17781
12	11540 - 11708
13	23768 - 37371
14	1800 - 12879
15	9616 - 19177
16	56 - 36998
17	11255 - 11436
18	1966 - 35918
19	10236 - 10250
20	15064 - 45178

Table 4.13: The table shows the last time interval where the *indexed* optimiser's (when run concurrently with the p-learner) current policy estimates converged to the true optimal policy before the computing stopped. The parameters used were $\mu = 20000$, $\sigma = 100$, $\Delta = 0.01$, and $T = 80000$ with seeds labelled 1 to 20. This was typical of any parameter set considered in our sequence of simulations.

In Table 4.12 for the parameter sets corresponding to $\mu = 20000$ we can see that all the optimal actions were found in all of the states, except state 4. Table 4.12 shows that in all the states bar state 4 the results are comparable to the known P case presented in Section 4.9, apart from the fact that the stopping time

in this case is considerably larger. Oscillating between the optimal action and the suboptimal action in state 4 was a common feature in e very parameter set and seed considered, because learning about all of the state transition probabilities connected to a state where the optimal Q-values are extremely close takes time. We predicted this may happen in Section 4.2. The reason the current optimal action estimates converged to th e true optimal action estimates in the higher valued states was not because the p-learner sampled these states a sufficient amount of times , as shown in the previous section, it was because the performance depended strongly on the numerical paramters used for the model. It was because the optimiser was able to sample these states and in each case the optimiser was fortunate in that the Q-values, although far from the true values for each state-action pair, were such that the current action estimates were the true optimal actions. In this and subsequent sections we will see that not observing the higher valued states in the p-learner will prove costly in the long run.

In Section 4.9 we saw that when the *indexed* optimiser was run on its own it found the true solutions at approximately the same time step, irrespective of the seed used. Looking at Table 4.13 above we can see that this was clearly not the case, when the *indexed* optimiser was run concurrently with the p-learner. Note that the lower limit reports the step time when the current policy estimate last oscillated from a suboptimal policy to the optimal policy, whereas the upper limit reports the step time at which computing stopped. Table 4.13 shows that both limits across each seed vary tremendously. This was true of all parameters sets and seeds considered.

In the last chapter we learnt that the variation in the lower limit over different seeds was caused by stochasticity owing to the way we chose actions, in a state which had Q-values that were extremely close to each other. An example of this in the last chapter was state 8, an example in this chapter is state 4. However, in this case both the limits vary because we did not gain good enough estimates of the state transition probabilities, not just in state 4 but in all of the higher valued

states. This is evident when we look at Table 4.11 presented in the last section. In the last section we saw that the p-learner hardly visited the higher valued states at all. Note in Table 4.13 only three out of the twenty seeds stopped with a suboptimal policy. They were seeds labelled 1, 11 and 18. However, this sort of performance might well depend strongly on the numerical parameters used for the model and not on us getting good estimates of the state transition probabilities, especially in the higher valued states. This again was typical of all the parameter sets and seeds considered in this sequence of simulations.

Figure 4.1 shows that all the current policy estimates settled down to a sub-optimal policy in all the parameter sets corresponding to $\mu = 10000$ and 20000, bar parameter set $\{\mu = 10000, \sigma = 400, \Delta = 0.01\}$. On the other hand, Figure 4.2 shows that all the current policy estimates settled down to the true optimal policy in all of the parameter sets corresponding to $\mu = 5000$, bar parameter set $\{\mu = 5000, \sigma = 100, \Delta = 0.01\}$. In addition to this the figures show that they all stopped at different time steps. However, we have already established that even though in some cases the current policy estimate converged to the true optimal policy, the results can be somewhat misleading. This is because the system was using the wrong state transition probabilities in the higher valued states.

In conclusion, if it is impossible to sample every state in a system a sufficient amount of times, irrespective of the learning parameters used, then recording times at which the current policy estimates settles down to the true optimal policy is not as useful as it might appear. This will become apparent when, in later sections, we look at the other convergence criteria.

4.11 Running the indexed optimiser concurrently with the p-learner - Convergence of the current policy estimate

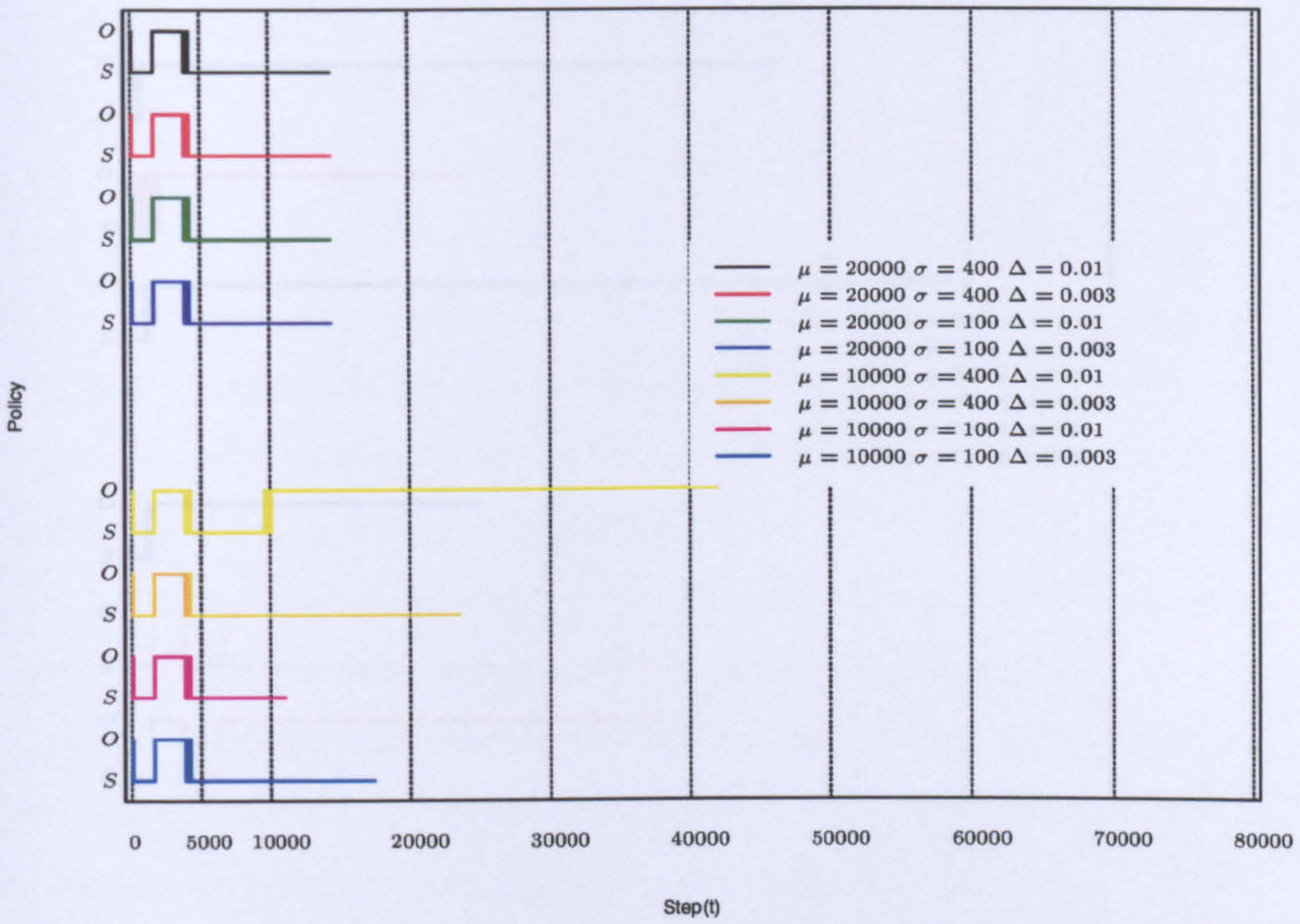


Figure 4.1: The graph above shows us the time intervals when the *indexed* optimiser's (when concurrently run with the p-learner) current policy estimates converged to the true optimal policy (O) and a suboptimal policy (S) respectively for the different combinations of parameters $\mu = 10000, 20000$, $\sigma = 100, 400$ and $\Delta = 0.003, 0.01$ and $T = 80000$ for seed 1. We can see in all parameter sets the current policy estimate oscillated between the optimal policy and a suboptimal policy.

4.11 Running the indexed optimiser concurrently with the p-learner - Convergence of the current policy estimate

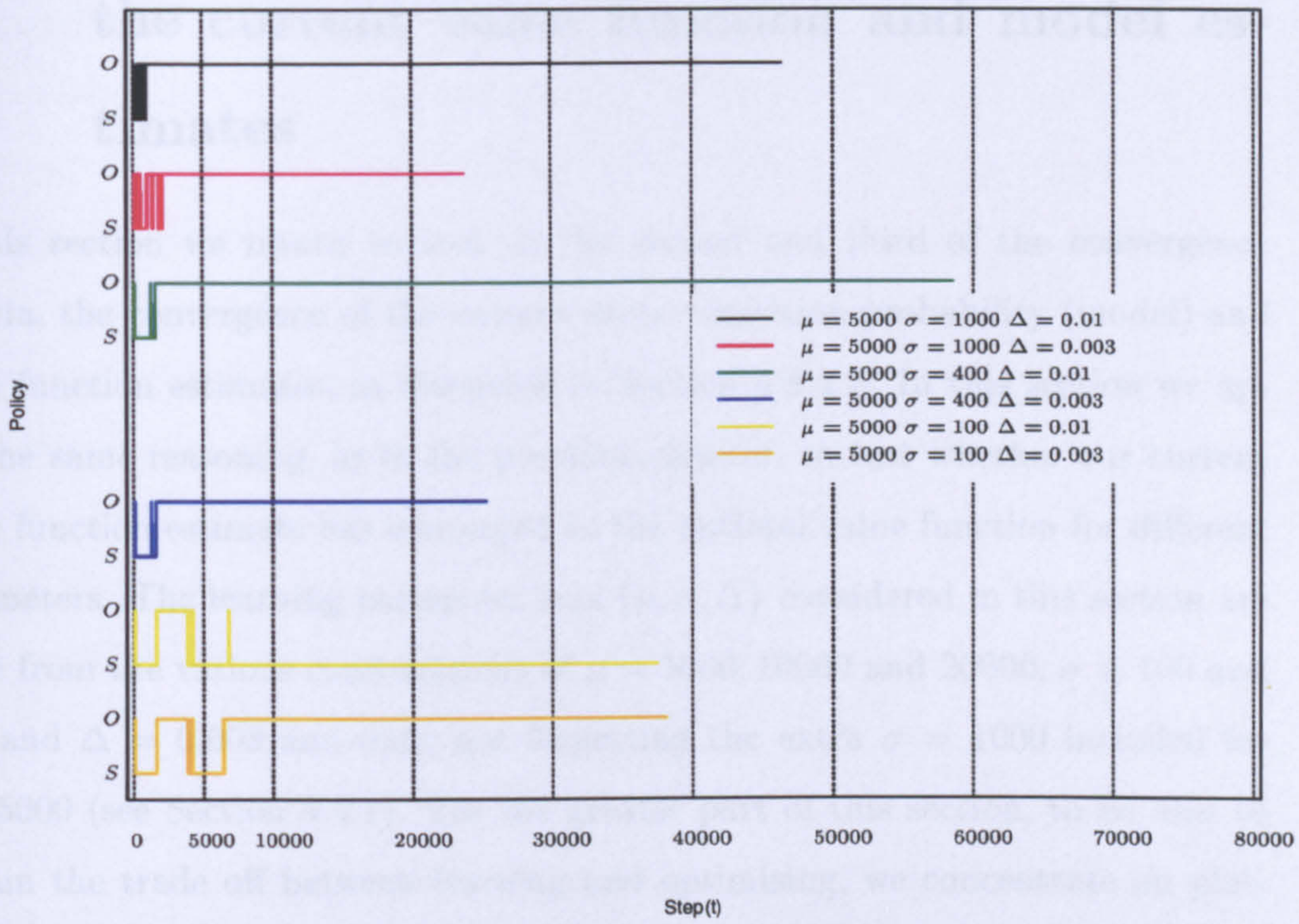


Figure 4.2: The graph above shows us the time intervals when the *indexed* optimiser's (when concurrently run with the p-learner) current policy estimates converged to the true optimal policy (O) and a suboptimal policy (S) respectively for the different combinations of parameters $\mu = 5000$ $\sigma = 100, 400, 1000$ and $\Delta = 0.003, 0.01$ and $T = 80000$ for seed 1. We can see in all parameter sets the current policy estimate oscillated between the optimal policy and a suboptimal policy.

4.12 Running the indexed optimiser concurrently with the p-learner - The convergence of the current value function and model estimates

In this section we return to look at the second and third of the convergence criteria, the convergence of the current state transition probability (model) and value function estimates, as discussed in Section 4.8.4.2. In this section we apply the same reasoning, as in the previous chapter, to test whether our current value function estimate has converged to the optimal value function for different parameters. The learning parameter sets $\{\mu, \sigma, \Delta\}$ considered in this section are made from the various combinations of $\mu = 5000, 10000$ and 20000, $\sigma = 100$ and 400, and $\Delta = 0.003$ and 0.01; not forgetting the extra $\sigma = 1000$ included for $\mu = 5000$ (see Section 4.4.1). For the greater part of this section, to be able to explain the trade off between learning and optimising, we concentrate on plotting the current estimates of the true state transition probabilities against the current value function estimates. In this section both single and multiple seeds are considered.

Figure 4.3 below demonstrates the performance of the method by looking at the evolution of the two criteria using the learning parameter set $\{\mu = 20000.0, \sigma = 100.0, \Delta = 0.01\}$, for two different seeds. We chose this parameter set because it typifies the problems in the performance of the *indexed* optimiser and p-learner when they are both run concurrently. Figure 4.4 on the other hand compares the method's qualitative performance of the errors in the current value function against time, with that of the optimiser when run on its own and the Pre-Jacobi method. Figure 4.5 plots the end points of the 20 different runs for parameter

sets corresponding to $\mu = 20000$ and $\mu = 10000$, to see if the two runs in Figure 4.3 are representative realisations of the process. Finally, we end this section by looking at Figure 4.6 to observe whether some states have larger average biases and standard deviations than others. Note in this section we have only plotted figures for the parameter sets corresponding to $\mu = 10000$ and $\mu = 20000$ because *they exemplify the typical problems encountered* when the *indexed* optimiser and p-learner are run concurrently.

To compare the two criteria we plotted the root mean square error of the current value function estimates for all $i \in S$, denoted by $E_{\tilde{v}}(t)$, against the root mean square error of all the current state-transition probability estimates for all $i, j \in S$ and $u \in U(i)$, denoted by $E_{\hat{p}}(t)$, parameterised at both time t and at log to the base 10 where,

$$E_{\tilde{v}}(t) = \sqrt{\left\{ \sum_{i \in S} \{v(i) - \tilde{v}_t(i)\}^2 \{|S|\}^{-1} \right\}}, \tag{4.12.17}$$

and

$$E_{\hat{p}}(t) = \sqrt{\left\{ \sum_{i \in S} \sum_{j \in S} \sum_{u \in U(i)} \{p_{ij}(u) - \hat{p}^t_{ij}(u)\}^2 \{46\}^{-1} \right\}}. \tag{4.12.18}$$

In $E_{\hat{p}}(t)$ we have made a slight change to the definition used in the previous chapter, to compensate for the structure of the problem. Instead of dividing by $|S|^2|U|$ as in Equation (3.11.6) in the fully connected case, in Equation (4.12.18) we divide by 46 because in the "sparsely connected" case there are only 46 state transition probabilities of interest. The others we know in advance and they are all set to zero.

In Section 3.11.4 we investigated the consistent trends in the performance of the method, looking at the evolution of the current value function and model estimates. We noted that to begin with the method learnt more about the true optimal value function, without learning too much about the true model. It is for this reason that even though the error in the current value function estimate

converged, it converged to something that was wrong. This is because the current policy estimate converged to a suboptimal policy. However, when the p-learner learnt more about the true state transition probabilities, the error in the current value function estimate oscillated depending on the policy taken. Subsequently as t increased the p-learner focused on the current action estimates, until eventually the error in the current value function estimates converged to zero.

Figure 4.3, on the other hand, shows us that this was not necessarily the case when the *indexed* optimiser and p-learner were applied to this "sparsely connected" system. It is true that to begin with the error in the value function estimate decreased relative to the error in the model, but unlike in the fully connected case the error in the current value function estimate did not converge to zero as t increased. This was true of all parameter sets and seeds used in this sequence of simulations, not just the parameters and seeds illustrated in Figure 4.3. This was because the p-learner could not observe the higher valued states as shown in Table 4.11 above. Note that if the p-learner cannot observe the higher valued states then it cannot learn about true state transition probabilities in these states. Consequently, the current value function converged to a point using the wrong model, until eventually the difference in the current value function estimate from one time step to another was negligible. As a result the computing was ultimately forced to terminate. We recall from Tables 4.12 and 4.13 that computing stopped after 14023 and 62532 iterations for seeds 1 and 2 respectively. If we now compare the error in the value function estimate for $t = 100$ with the stopping time for both seeds, we can see that the difference in the two for each seed is negligible.

To compare the performance of the *indexed* optimiser when concurrently run with the p-learner, with that of the optimiser when run on its own and the Pre-Jacobi method, we then looked at the error in the current value function estimate with respect to time t. As in Section 3.11.5 we slightly changed the notation of Equation (4.12.17) to report the estimates for the two other methods

listed above. Thus by substituting $\widehat{v}$ and v for $\widetilde{v}$ in Equation (4.12.17) for the indexed optimiser when run on its own and the Pre-Jacobi method respectively, we now have statistics that denote the root mean square error of the current value function estimate. Again we used the learning parameter set $\{\mu = 20000, \sigma = 100, \Delta = 0.01\}$ to demonstrate the comparison, but they are typical of results gained using any other learning parameters and seeds.

Figure 4.4 below shows as expected that the Pre-Jacobi was the most dominant method, followed closely by the optimiser when run on its own. We recall that the Pre-Jacobi, the indexed optimiser run on its own and the indexed optimiser run concurrently with the p-learner stopped after 46, 461 and 14423 iterations respectively. However, we must state that in the methods where the system parameters were known $E_v(t)$ (for $v = v$ and $v = \widehat{v}$) converged to zero every time, whereas in the unknown P case $E_{\widetilde{v}}(t)$ did not equal zero irrespective of the learning parameters and seeds used. This is because the p-learner was unable to observe the higher valued states (see Table 4.11). Note when the optimiser was run on its own, unlike in the fully connected case where the error in the current value function estimate monotonically decreased as t increased, in this case the error either monotonically decreased or was constant as not all of the states were fully connected.

To investigate the outcome of each learning parameter set and seed, we plotted their corresponding end points. Note that the end points in this section are defined slightly differently from those in Section 3.12. In this section as each run has a certain stopping time we define the end points to be $\log_{10}\{E_{\widetilde{v}}(H_N)\}$ and $\log_{10}\{E_{\hat{p}}(H_N)\}$, where H_N is the stopping time for seed N. In Section 3.12 we saw that the p-learner learnt more about the overall model for parameter sets corresponding to $\mu = 20000$ and $\mu = 10000$ compared with $\mu = 5000$. However, in the sparsely connected case this was not the case. The graph for the end points plotted for the learning sets corresponding to $\mu = 5000$ is omitted from this section, but it is very similar to the one illustrated in Figure 4.5. In Figure 4.5

we can see that, irrespective of the learning parameter sets used, the end points are more tightly clustered than in the case of the fully connected case. This is apparent through examining not only the axis corresponding to the error in the current value function estimate, but also the error in the true state transition probabilities. This again is a consequence of not observing the higher valued states in the p-learner (see Table 4.11).

To investigate whether some current value function estimates in some states were better estimates of their corresponding true values than others, we plotted the average standard deviation $\widetilde{v}_{H_N}(i)$ against the average bias of $\widetilde{v}_{H_N}(i)$ (over the 20 different seeds) for each parameter set and state $i \in S$. In Figure 4.6 observations for each state are labelled from 0 to 11, and each observation for each learning parameter set is labelled in one colour. Figure 4.6 demonstrates by example the consequences of not observing the higher valued states in the p-learner in the learning parameter sets corresponding to $\mu = 20000$ and $\mu = 10000$. The results we present below are representative of the ones obtained using learning parameter sets corresponding to $\mu = 5000$.

When we introduced similar graphs in the previous chapter (Section 3.13) we saw that the principal features were the observations in states where the p-learner did not sample the true optimal actions, since each state was visited fairly often. However, in this sparsely connected system the principal features are the states the p-learner did not observe very often, or at all. They are chiefly the higher valued states.

We can see in Figure 4.6 that the lower valued states, states $\{0, 1, 2, 3, 4\}$, both have very small biases and standard deviations. On the other hand, the states in the transient state space, states $\{5, 6, 7, 8, 9, 10, 11\}$ (the higher valued states), have small biases but high standard deviations. In Table 4.10 we saw that the optimiser sampled each state uniformly, whereas Table 4.11 shows that the p-learner observed states according to a distribution of a geometric nature. It is for these reasons alone that the higher the value of the state the larger the

biases and standard deviations become.

The optimiser sampled each state fairly often so that the current value functions, in each case, converged to a point which was not a long way short of the true values in each state. In the case of the higher valued states this was probably caused by the discrepancies in the estimated model compared with the true model. However, because the p-learner sampled the lower valued states fairly often, their corresponding standard deviations were small and the observations for the same states were tightly clustered. On the other hand, because the p-learner hardly observed the higher valued states their corresponding standard deviations were large by comparison, and in many cases the observations for the same state were more scattered. Note that the p-learner barely visited states 10 and 11. Therefore the current value functions corresponding to states 10 and 11 had the highest biases, but they were clustered together more closely than any of the other higher valued states because their state transition probabilities (over each learning parameter set) were approximately equal.

In conclusion, the evidence we have presented in this section shows that if not all of the states in the p-learner are observed fairly often, then the estimates of the optimal value function suffer as a result. When we looked at the times at which the current policy estimates equalled the true optimal policy, the consequences of not observing all the states in the p-learner were not so visible. We have also shown that the p-learner encountered difficulties with the problem's structure irrespective of the learning parameters used. In this case new methodology must be developed to deal with systems that are sparsely connected. In the final section of this chapter we introduce two new ideas used in the p-learner that deal with sparsely connected systems.

4.12 Running the indexed optimiser concurrently with the p-learner - The convergence of the current value function and model estimates

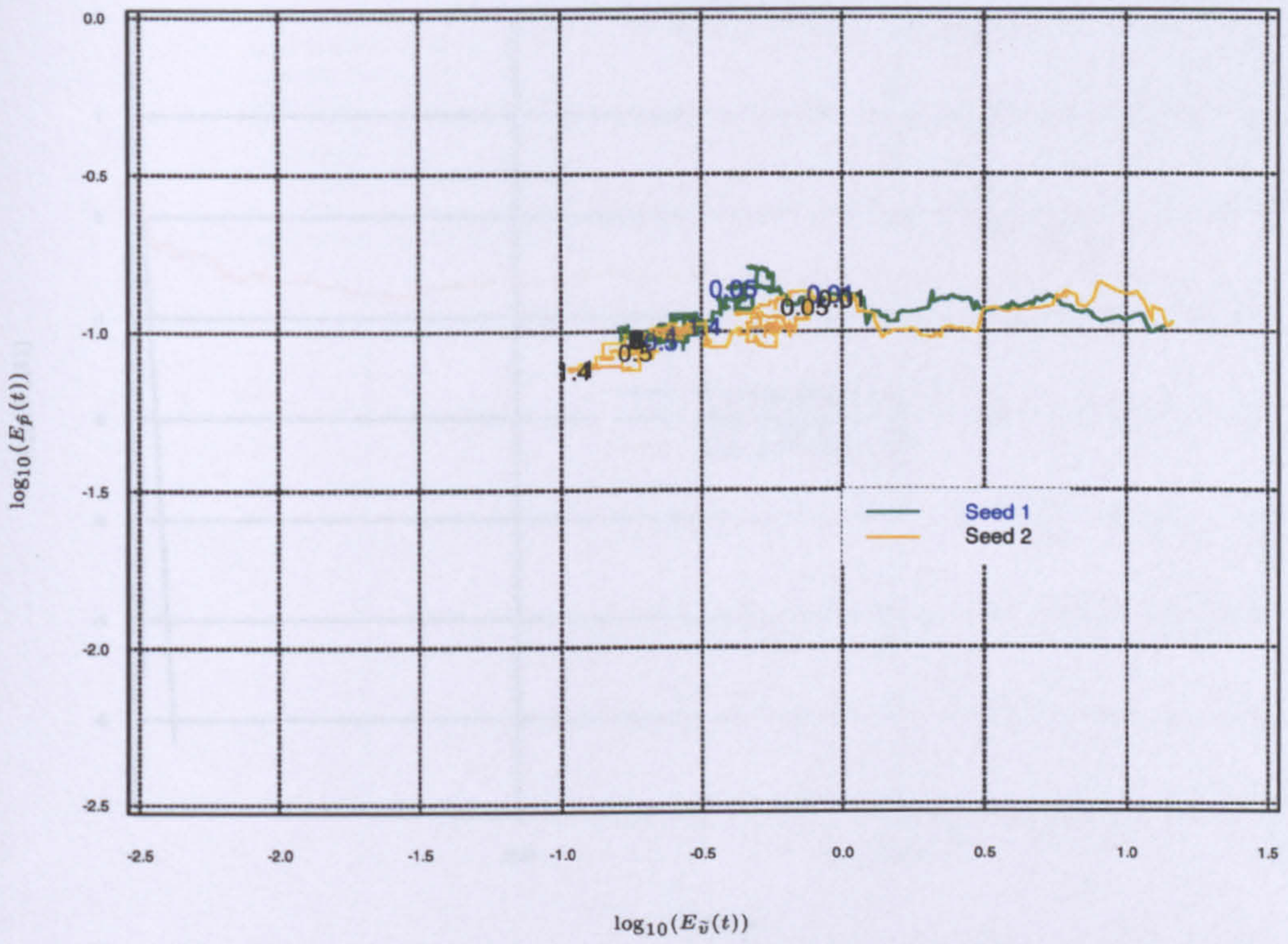


Figure 4.3: The graph above is a plot of $\log_{10}(E_{\tilde{v}}(t))$ against $\log_{10}(E_{\hat{p}}(t))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; where $\mu = 20000.0$, $\sigma = 100.0$, $\Delta = 0.01$ and $T = 80000$. Twenty different seeds were used in all. We present results taken from two seeds to give us a good illustration of the method's performance. Tick marks were plotted for each of the two seeds at 100, 500, 5000, 10000, 14000 iterations, ; labelled 0.01, 0.05, 0.5, 1 and 1.4. However, for seed 2 we plotted tick marks at 20000 iterations and every 10000 iteration thereafter until computing terminated, labelled from $2, 3, 4, 5$, and 6.

4.12 Running the indexed optimiser concurrently with the p-learner - The convergence of the current value function and model estimates

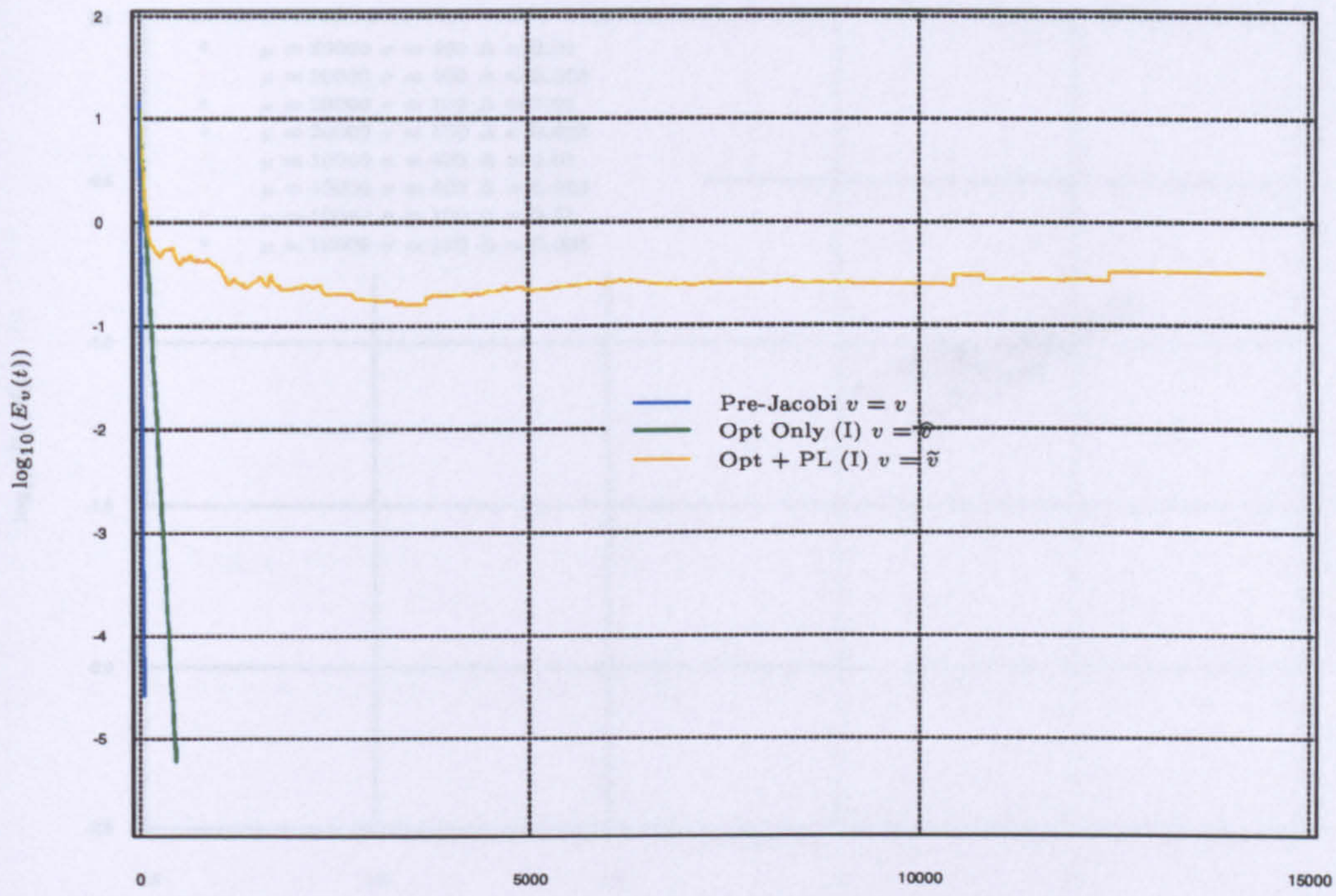


Figure 4.4: The graph above is a plot of root mean square error of current value function estimate at $\log_{10}$ against step time t using parameters $\mu = 20000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ for seed 1. We wanted to compare the error in the value function for the 3 different methods; the optimiser run on its own (Opt Only), the optimiser run concurrently with the p-learner (Opt + PL) and the conventional Pre-Jacobi method. We plotted the errors in the value function for these particular parameters because qualitatively speaking they display roughly the same typical behaviour over the 3 different methods as in any other set of parameters.

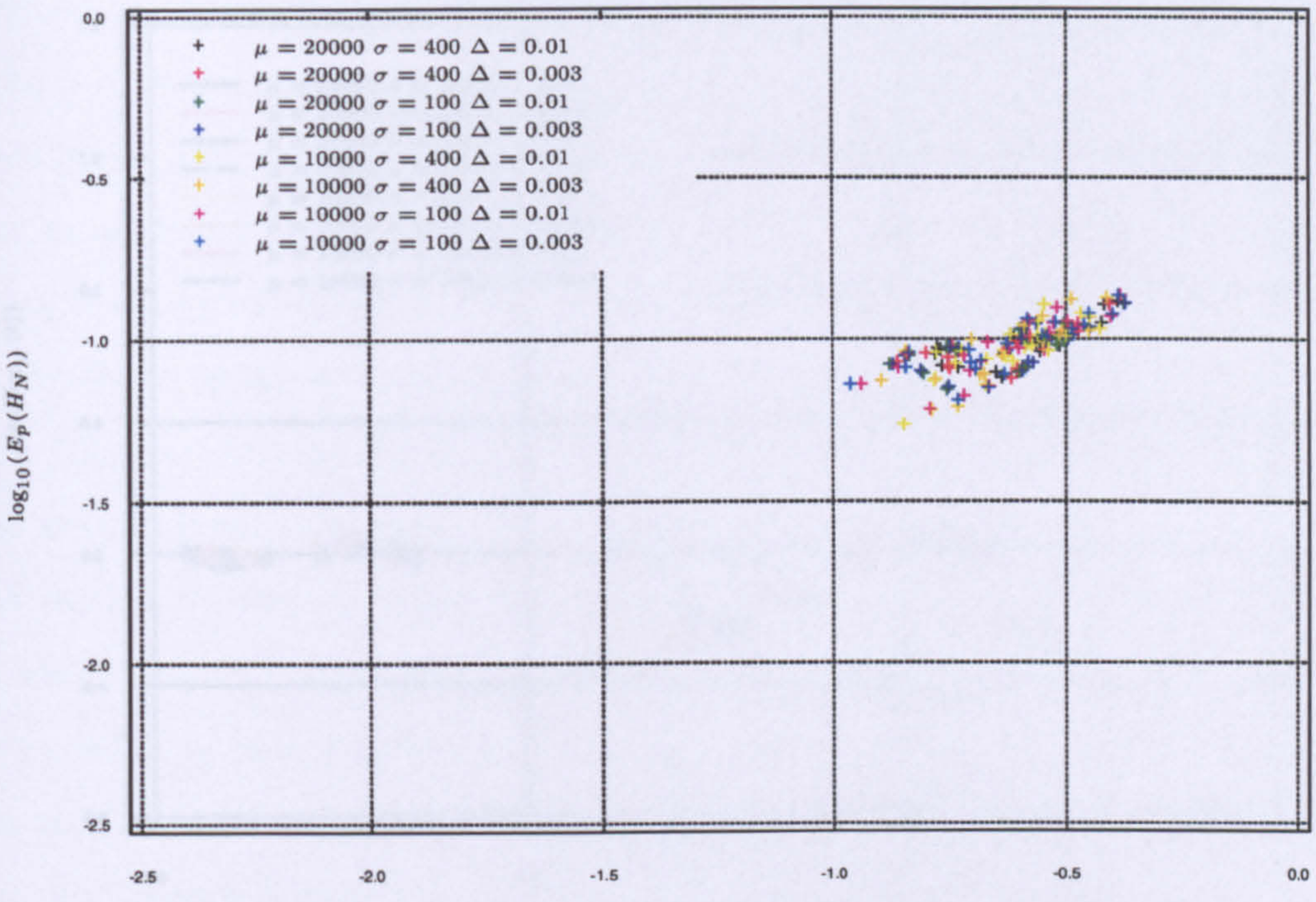


Figure 4.5: The graph above is a plot of $\log_{10}(E_v(H_N))$ against $\log_{10}(E_p(H_N))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for the different combinations of $\mu = 10000, 20000.0$, $\sigma = 100.0, 400.0$, and $\Delta = 0.003, 0.01$. The graph illustrates the position of the end points of all 20 simulations using different seeds for each parameter set.

4.12 Running the indexed optimiser concurrently with the p-learner - The convergence of the current value function and model estimates

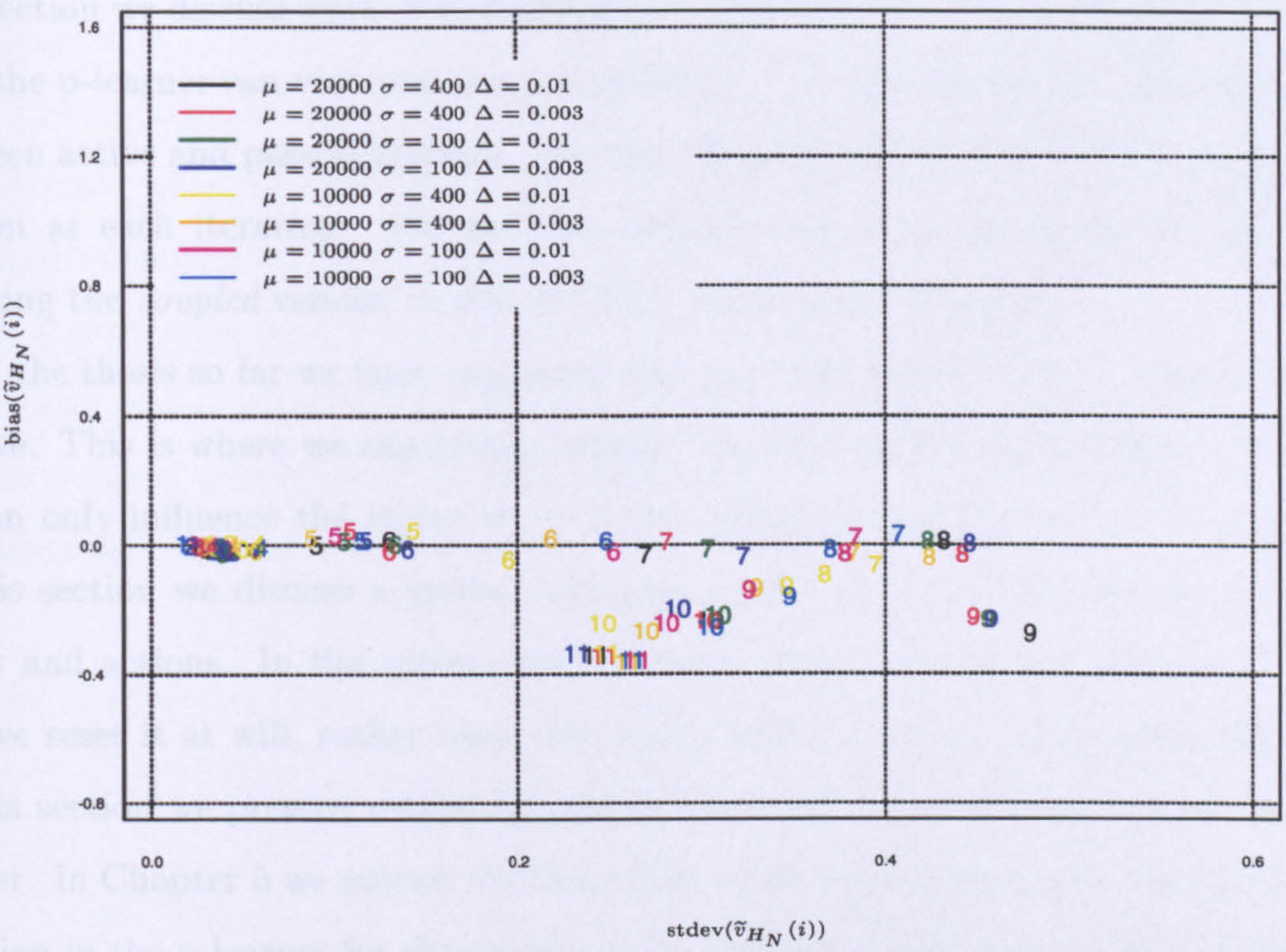


Figure 4.6: The graph above is a plot of the average standard deviation of $\widetilde{v}_{H_N}(i)$ against the average bias of $\widetilde{v}_{H_N}(i)$ defined for all states $i \in S$ at time T for the parameter sets considered for $\mu = 20000$ and $\mu = 10000$. Each observation consists of results taken from the 20 simulations for each parameter set. We used one colour for each parameter set and each observation represents the value of each state.

4.13 Active and passive systems

This final part of the chapter discusses the *coupled* version of the methodology. In this section we discuss ways of making further modifications to our algorithm, so that the p-learner can visit states more uniformly. We first discuss the difference between active and passive systems. We then describe the function of the *coupled* version at each iteration. We end this chapter reporting results gained from applying the *coupled* version to this machine replacement problem.

In the thesis so far we have employed systems using "black boxes", which are passive. This is where we can always observe the state of the (real) system, but we can only influence the states we go to by taking actions (see Section 2.3.1). In this section we discuss a system which is active, where we can choose both states and actions. In the active case we choose each state in the "black box" and we reset it at will, rather than only being able to observe states internally. In this section we present results by coupling the *indexed* optimiser with the p-learner. In Chapter 5 we extend the idea of the index method to a more sensitive criterion in the p-learner for the passive case. We have formulated an idea using indices in the p-learner similar to those defined in the optimiser. This method is a general way of observing states in the p-learner through index numbers, so that we are tracking the process at each iteration rather than relying on knowing the structure of the problem. Future work might well investigate this approach further.

The mechanics of the *coupled* version are very similar to that of the *indexed* version, but its applicability is somewhat different. This is because, in the *coupled* version, the mechanism we use for learning about the true state transition probabilities is "resetable" and is under our control. When the optimiser and p-learner are coupled we use the index (state) method in the optimiser to choose which state to go to in the p-learner, and we use Method 3 to choose actions. In the system the optimiser gives its current state i to the p-learner and the

p-learner then uses this state to choose an action u, according to the exploration function defined in Section 3.3.4. Having identified the current state-action pair (i, u) the p-learner then chooses a state j, as illustrated in Figure 4.7 below. The transition from i to j given action u is recorded, and the state j is then disregarded. Thus, at each iteration the "black box" is reset because at each iteration the p-learner uses the next state given to it by the optimiser. This enables the p-learner to sample states uniformly. The course of reseting the black box each time is a legitimate one. For each state i we are trying to learn about the distribution over each j. Therefore a variable j of size one from this distribution can be accumulated with other data of this kind, to help us estimate the true state transition probabilities.

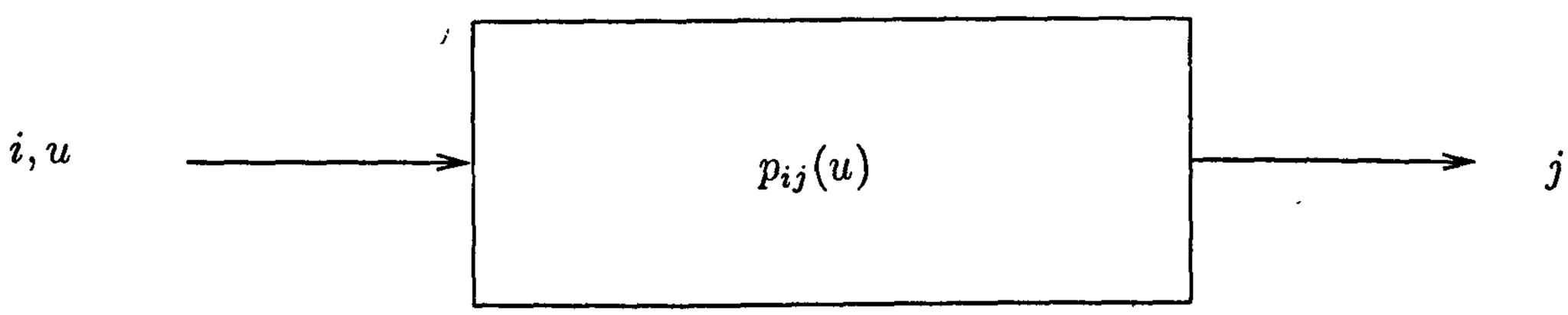


Figure 4.7: The diagram above illustrates how the p-learner simulates the real state transition probabilities by choosing both the system's current state $i \in S$ and $u \in U(i)$ and issuing them to the Black Box to get an update of the process $j \in S$.

In the remaining section we present results obtained running the coupled version of the *indexed* optimiser concurrently with the p-learner, to see how well the methods cope with estimating the true optimal value function $v^*(i)$ and the true optimal policy $\pi^*(i)$ for each state $i \in S$. We present results to see if the *coupled* version is successful in estimating the optimal solutions of this particular case study. The purpose of presenting these results is not to show which learning parameters are best, but to show that the methodology works when applied to sparsely structured problems.

The learning parameter sets for $\{\mu, \sigma, \Delta\}$ considered are the ones got from $\mu = 5000, 10000$, and 20000, $\sigma = 100$ and 400, $\Delta = 0.003$ and 0.01, with an additional $\sigma = 1000$ included for $\mu = 5000$. We recall that μ denotes the amount of time allocated to learning, σ denotes how quickly we move from total sampling to focused sampling, and Δ determines the level of discrimination between actions $u \in U(i)$ in each state $i \in S$. Note that the smaller the value of σ the quicker the transition from total sampling to focused sampling, and the greater the value of Δ the larger the level of discrimination between actions in each state. Note that when we analysed the results for the current policy estimates in Section 4.11, problems in the p-learner were less apparent than when we analysed figures that incorporated results for the current value function and model estimates for multiple runs. Since we only want to demonstrate that the coupled methods cope with sparsely connected systems, this section only considers multiple seeded graphs.

By plotting the end points of the 20 different runs for learning parameter sets corresponding to $\{\mu = 10000, 20000\}$ and $\{\mu = 5000\}$ in Figures 4.8 and 4.9 below respectively, we illustrate the method's performance of reducing the error in the current value function estimate and the current model. These figures show that the learning parameter sets used have a profound effect on the method's performance. They are then compared with the graphs presented when the *indexed* optimiser and p-learner were run independently (Section 4.12). Figure 4.10 shows how good the final average estimates were, over the 20 different seeds, by plotting the average standard deviation $\widetilde{v}_{H_N}(i)$ against the average bias of $\widetilde{v}_{H_N}(i)$ (over the 20 different seeds) for each parameter set and state $i \in S$. This graph is also compared with results obtained in Section 4.12.

Figures 4.8 and 4.9 show that there are significant differences in the end points. In fact these differences are similar to the ones presented in the previous chapter. In Section 4.12 we saw that in Figure 4.5 the end points for the learning parameter sets were tightly clustered, and they are situated more towards the top right hand side of the graph. The same was also true of the learning parameter

sets corresponding to $\mu = 5000$. In fact the two graphs looked very similar. This was because the p-learner could not observe the higher valued states, and as a result the current value estimates converged to a point until computing was forced to stop. However, Figures 4.8 and 4.9 show that the higher the value of μ, the more the p-learner learnt about the overall model. This was a principal feature in the graphs presented in the previous chapter. It is because both the p-learner and optimiser were able to visit each state fairly often. However, we must note that hardly any of the runs stopped before time T. This shows that the cost of not knowing the true state transition probabilities is fairly high, since when we know the probabilities the methods stopped at time $t << T$ with optimal solutions.

To investigate whether some states had on average larger standard deviations and biases than others, we present the results from the learning parameter sets corresponding to $\mu = 10000$ and 20000. We can see that unlike Figure 4.6, Figure 4.10 does not show any large standard deviations. This was because, this time, each state was visited uniformly in the p-learner. In fact, for $\mu = 5000$ it is true that there was no evidence of large standard deviations, and that they were no bigger than the ones reported for $\mu = 10000$ and 20000. This was due to the fact that in this problem $\mu = 5000$ was not too early a point to discriminate between actions. However, unlike in the previous chapter the p-learner did not have to learn about so many transition probabilities, therefore on average the results presented in Figure 4.10 are far better than those presented in Figure 3.28. However, although the results we present are good, the *coupled* version does not perform as well as methods that use the true state transition probabilities.

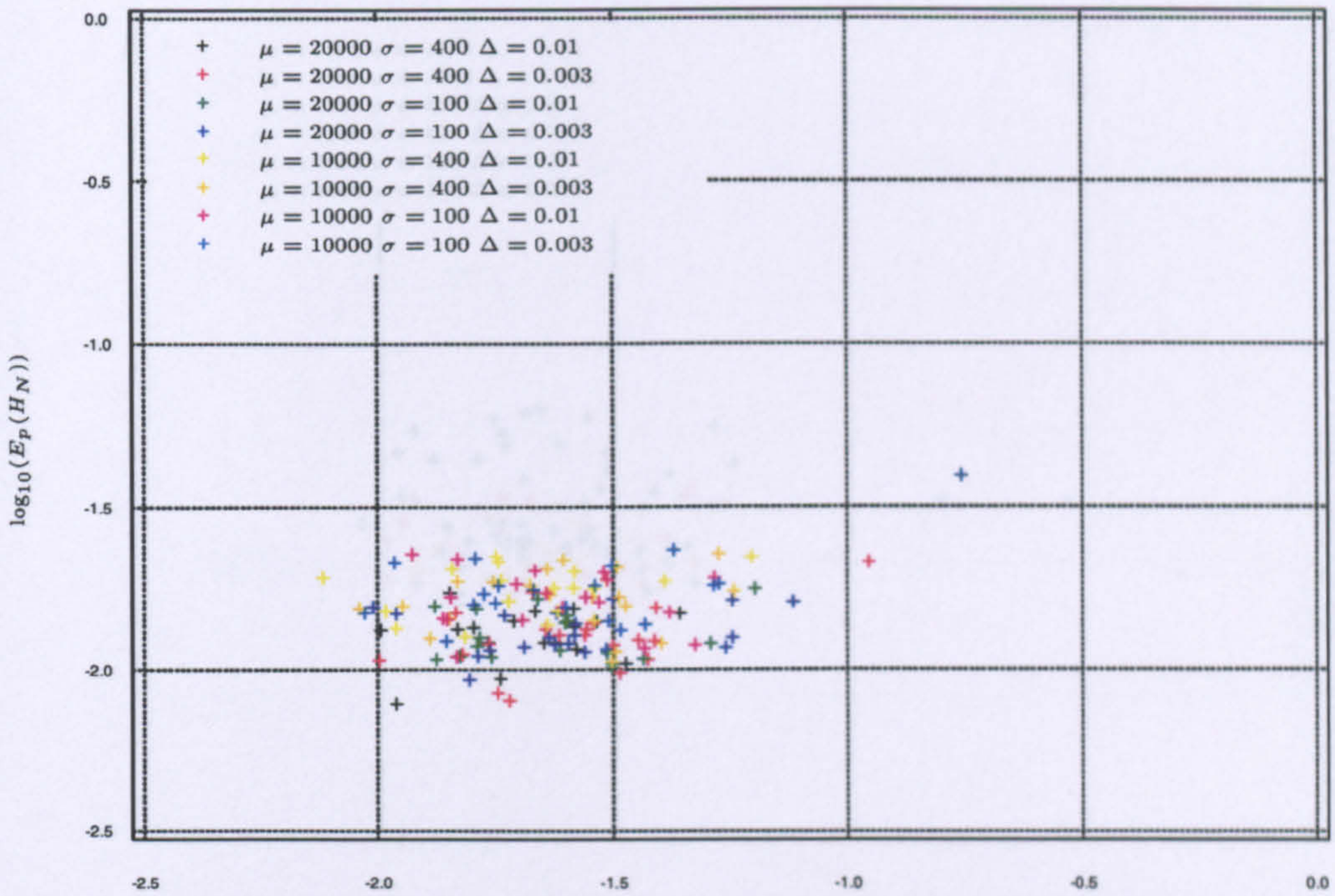


Figure 4.8: The graph above is a plot of $\log_{10}(E_v(H_N))$ against $\log_{10}(E_p(H_N))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for the different combinations of $\mu = 10000, 20000.0$, $\sigma = 100.0, 400.0$, and $\Delta = 0.003, 0.01$. The graph illustrates the position of the end points of all 20 simulations using different seeds for each parameter set.

4.13 Active and passive systems

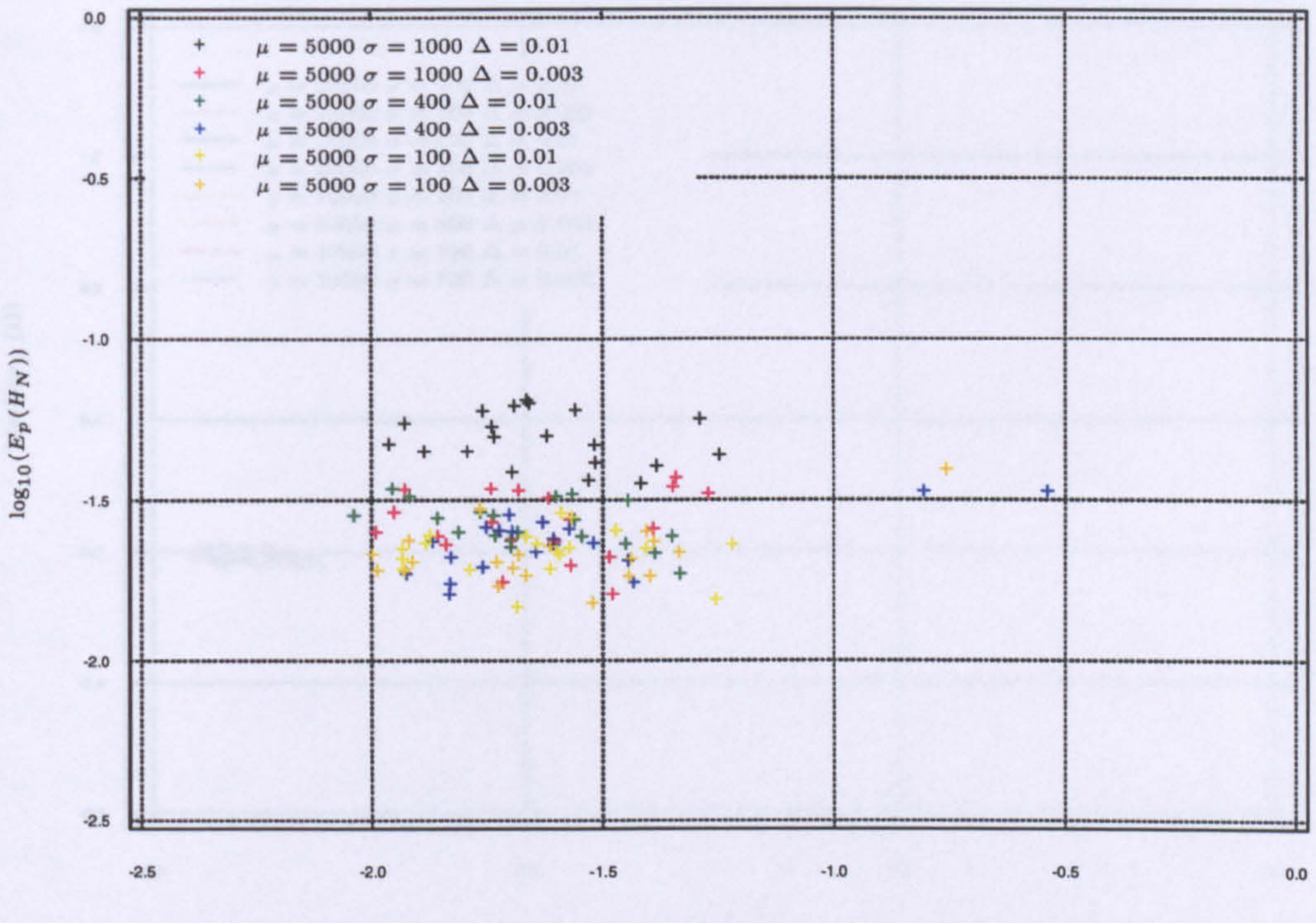


Figure 4.9: The graph above is a plot of $\log_{10}(E_v(H_N))$ against $\log_{10}(E_p(H_N))$ (parameterised by time t) defined for all states $i \in S$ and actions $u \in U(i)$; for the different combinations of $\mu = 5000.0$, $\sigma = 100.0, 400.0, 1000.0$ and $\Delta = 0.003, 0.01$. The graph illustrates the position of the end points of all 20 simulations using different seeds for each parameter set.

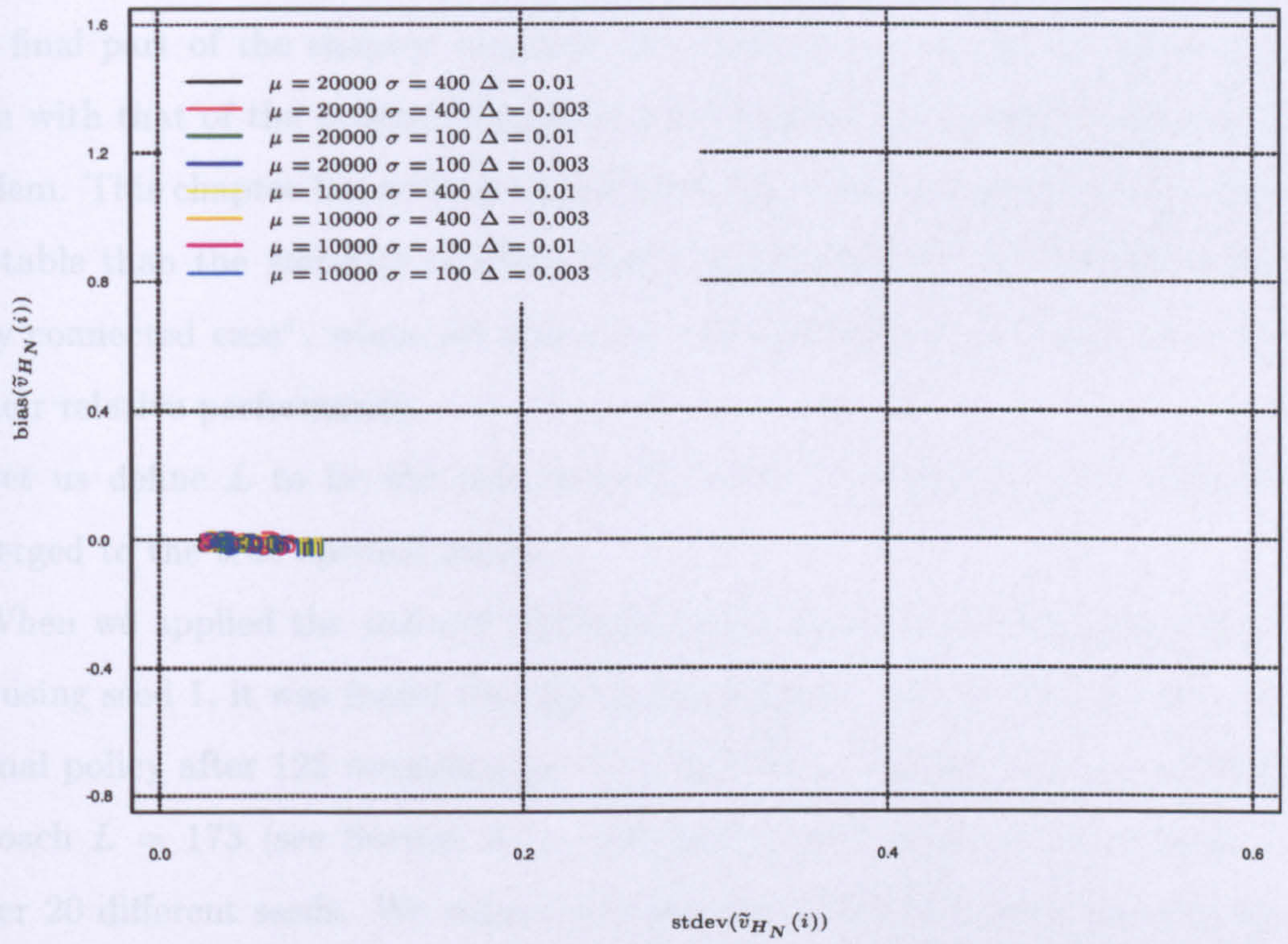


Figure 4.10: The graph above is a plot of the average standard deviation of $\widetilde{v}_{H_N}(i)$ against the average bias of $\widetilde{v}_{H_N}(i)$ defined for all states $i \in S$ at time T for the parameter sets considered for $\mu = 20000$ and $\mu = 10000$. Each observation consists of results taken from the 20 simulations for each parameter set. We used one colour for each parameter set and each observation represents the value of each state.

4.14 Further Comparison of the *Standard* and *Indexed* Optimisers

This final part of the chapter compares the performance of the *standard* algorithm with that of the *indexed* algorithm when applied to the "fully connected" problem. This chapter has already shown that, the *indexed* approach is far more adaptable than the *standard* version. The comparison of the algorithms in the "fully connected case", where all states are well determined, will give us a clue to their relative performance.

Let us define L to be the time step at which the current policy estimate converged to the true optimal policy.

When we applied the *indexed* optimiser on its own to the "fully connected" case using seed 1, it was found that the policy estimate settled down to the true optimal policy after 122 iterations i.e. $L = 122$. We recall that for the *standard* approach $L = 173$ (see Section 3.7). We then looked at the average value of L over 20 different seeds. We report that for the *indexed* optimiser the average value of L was 120, whereas the average value of L was nearly double that value (for all the different combinations of parameters for $\{\mu, \sigma, \Delta\}$) using the *standard* optimiser.

To compare the performance of current value function estimates, when running the optimisers on their own, we introduce a simple stopping criterion for the *standard* method which is similar to the one used for the Pre-Jacobi defined in Equation (4.9.16). The stopping criteria is set up so that the computing stops once the current value function estimate equals the true optimal value function in each state $i \in S$ to three decimal places. The stopping criterion in the *standard* optimiser is defined in Equation (4.14.19) below. It stops computing when,

$$|v_{t+1}(\cdot) - v_t(\cdot)| < 1 \times 10^{-8}. \tag{4.14.19}$$

We report for seed 1 that the *indexed* method stopped after 1209 iterations compared with 1821 iterations (for the parameter set $\{\mu = 20000, \sigma = 400, \Delta = 0.01\}$) using the *standard* version. Over the 20 different seeds the current value function converged, on average, after 1219 iterations for the *indexed* version and after 1647 iterations for the *standard* version, using the parameter set $\{\mu = 20000, \sigma = 400, \Delta = 0.01\}$; similar results were gained for other parameter sets.

In order to investigate whether the *indexed* method had difficulties finding optimal actions in certain states, we looked at the individual time steps where the current action estimates converged to true optimal actions in each state $i \in S$. The results recorded for *indexed* method using seed 1 are illustrated in Table 4.14.

State $i \in S$	The time intervals at which $\widehat{\pi}_t(i) = \pi^*(i)$ for each individual state $i \in S$
0	9 - 1209
1	10 - 1209
2	12 - 1209
3	4 - 1209
4	3 - 46 52 - 93 103 - 1209
5	5 - 1209
6	1 - 1209
7	7 - 1209
8	45 - 90 122 - 1209
9	6 - 1209

Table 4.14: The table shows us the time intervals at which the *indexed* optimiser chose each optimal action $\pi^*(i)$ in each individual state $i \in S$ using seed 1. These results are characteristic of the results gained using any seed. Note the stopping time at $t = 1209$.

They are characteristic of the results gained using any seed. Table 4.14 tells us that the *indexed* optimiser found all of the true optimal actions in each individual

state with ease apart from states 4, 5 and 8. This was to be expected after looking at the results gained for the *standard* optimiser, illustrated in Table 3.5. On the other hand, we note that the *indexed* optimiser out performs the *standard* optimiser. This is reflected both in the evolution of the current policy and in the value function estimates.

As the performance of the *indexed* optimiser is far superior to that of the *standard* optimiser when the state transition probabilities were known, we would

Seed	The last time intervals at which $\widetilde{\pi}_t(\cdot) = \pi^*(\cdot)$ for different seeds
1	16120 - 80000
2	24236 - 80000
3	5817 - 80000
4	169 - 80000
5	6702 - 80000
6	50152 - 80000
7	10228 - 80000
8	4905 - 80000
9	19811 - 80000
10	4809 - 80000
11	372 - 80000
12	2199 - 80000
13	40539 - 80000
14	1428 - 80000
15	4651 - 80000
16	25281 - 80000
17	2837 - 80000
18	12395 - 80000
19	19696 - 80000
20	96 - 80000

Table 4.15: The table shows us the last time interval where the *indexed* optimiser's (when run concurrently with the p-learner) current policy estimates converged to the true optimal policy for $\mu = 20000$, $\sigma = 400$, $\Delta = 0.01$, and $T = 80000$ using seeds labelled 1 to 20. This was typical of any parameter set used in our sequence of simulations.

expect the performance of the *indexed* optimiser to be at least as good as that

of the *standard* version when they are unknown. This is indeed what we find in practice. Table 4.15 above illustrates the typical problems encountered when the *indexed* optimiser was run concurrently with the p-learner. The table reports the last time intervals when the *indexed* optimiser's current policy estimates settled down to the true optimal policy using the parameter set $\{\mu = 20000, \sigma = 400, \Delta = 0.01\}$ and seeds labelled 1 to 20. We recall seeing similar results for the *standard* optimiser in Table 4.15. These similarities are due to the trouble the optimiser has in differentiating between the true optimal and the best suboptimal action in state 8 (because it takes time to gain good estimates of the state transition probabilities) and the stochasticity owing to the way we choose actions in the p-learner. For both optimisers, irrespective of the learning parameters and sets used, if the true state transition probabilities were known the current policy estimate converged to the true one and at roughly the same time.

The underlying deficiency in the p-learner also affects the optimiser's effectiveness in quantifying the current value estimate. Figure 4.11 below illustrates the optimisers ability to calculate the current value estimate. The graph plots the error in the current value function estimates for each method using Equation (3.11.5) with respect to step time t. This plot is again typical of any learning parameter set and seed used. We can see that for both methods the errors $E_v(t)$ do not converge to zero even for t equal to 80000 and that neither method out performs the other. This was not the case when the true transition probabilities were known, the *indexed* method being the far superior method.

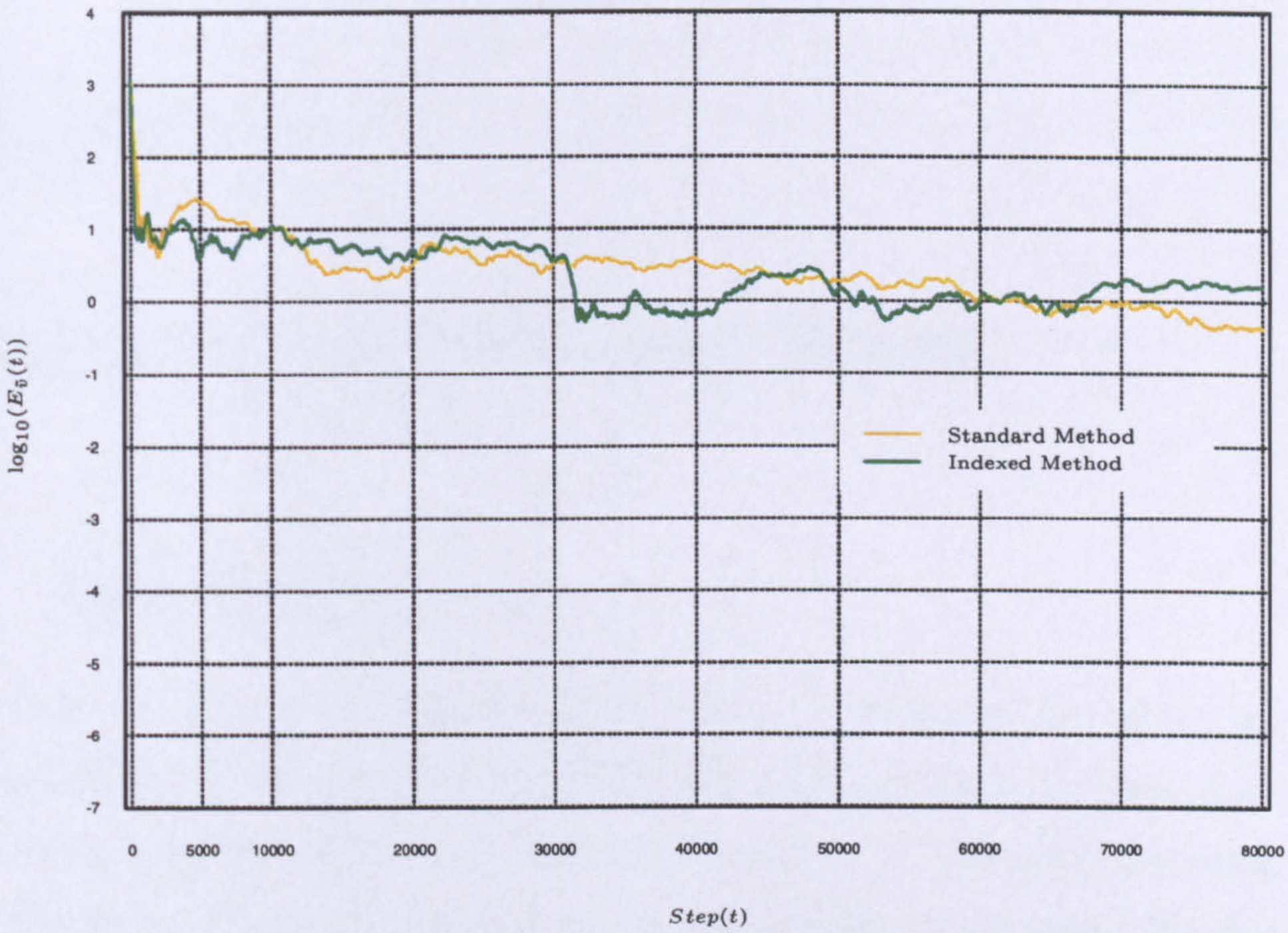


Figure 4.11: The graph above compares the performance of the *standard* optimiser with that of the *indexed* optimiser for the "fully connected" case illustrated in Chapter 3, when both optimisers are run concurrently with the p-learner. The graph shows the root mean square error of current value function estimate at $\log_{10}$ plotted against step time t using parameters $\mu = 20000.0$, $\sigma = 400.0$, $\Delta = 0.01$ and $T = 80000$ for seed 1. We plotted the errors in the value function for these particular parameters because qualitatively speaking they display roughly the same typical behaviour as in any other set of parameters considered.

Chapter 5

Ongoing and Future Work

5.1 Introduction

This chapter discusses ongoing and future work. Although we can prove the convergence of the estimates generated by the optimiser when the true state transition probabilities are known, future work needs to be addressed to proving the convergence of these estimates when the probabilities are unknown. We also propose future work on a new index method in the p-learner.

5.2 Proving Convergence using the Optimiser

5.2.1 Overview

This section looks at the convergence of the solutions generated by the optimiser over an infinite time period. The proof is based on the following assumption:

Assumption 1 each state $i \in S$ is visited infinitely often.

The optimiser randomly samples states in a manner correlated with a simulation of the process. The *standard* optimiser used in case study 1 chooses actions

in order to move to states, whereas the *indexed* optimiser used in case study 2 visits its states according to the probability distribution of the indices. For the current value function and policy estimates to converge to their corresponding optimal values, assumption 1 must hold true. The optimisers we have developed are not defined over an infinite time period. We can, however, extend the definition of the optimisers so that they are. Below we have indexed functions using step time t.

In order to expand our *standard* optimiser to run over an infinite simulation horizon, we slightly change the probability distribution $q_{ij}(u)$ used to choose states. As we recall, this method is described in Section 2.2.2. We can extend it to an infinite horizon by specifying that, when in state $i_t = i$ at time t an action $u_t = u$ is chosen, the next state $i_{t+1} = j$ is chosen with probability,

$$\begin{aligned} q_{ij}(u) &= p_{ij}^{1/T_t}(u) \left\{ \sum_{l \in S} p_{il}^{1/T_t}(u) \right\}^{-1} \quad \text{if} \quad t \leq T \quad \text{and,} \\ &= p_{ij}(u) \quad \text{otherwise.} \end{aligned}$$

where the function T_t is the temperature schedule at time t and is defined at $t = 1, 2, \ldots, T$ in terms of the finite horizon T by,

$$T_t = \{1 + \exp(4 - 8t/T)\}.$$

We recall in the *indexed* optimiser that computing stops once,

$$\sum_{j \in S} \tau_j < D,$$

where τ_j is an index defined for each state $j \in S$ and D is the threshold parameter set at the start of each simulation. To modify the *indexed* version of the optimiser so that it could run for an infinite amount of time, we need only omit this stopping criterion.

In the sections below we prove the convergence of the estimates, generated by these modified versions of the optimisers.

5.2.2 The Proofs

At each time point the optimiser visits a state i_t and updates the current value function and policy estimate for that current state. Since the optimiser when run on its own knows all the immediate costs $c_i(u)$ and the true state transition probabilities $p_{ij}(u)$ for all $i, j \in S$ and $u \in U(i)$, it does not matter what random sampling routine is used to update the current value function estimate $\widehat{v}_t(\cdot)$ and the current policy estimate $\widehat{\pi}_t(\cdot)$. We show that, as long as each state is visited infinitely often, $\widehat{v}_t(\cdot)$ and $\widehat{\pi}_t(\cdot)$ will tend to $v^*(\cdot)$ and $\pi^*(\cdot)$ respectively as $t \to \infty$, i.e $\|\widehat{v}_t - v^*\| \to 0$ where $\|\widehat{v}_t - v^*\| = \sup_j |\widehat{v}_t(j) - v^*(j)|$.

Lemma 1

Let i_t be the current state updated at time t then,

(i) $|\widehat{v}_{t+1}(i_t) - v^*(i_t)| \leq \alpha \|\widehat{v}_t - v^*\|$,

(ii) $|\widehat{v}_{t+1}(j) - v^*(j)| = |\widehat{v}_t(j) - v^*(j)| \quad \text{for all} \quad j \neq i_t$,

(iii) $\|\widehat{v}_s - v^*\| \leq \|\widehat{v}_t - v^*\| \quad \text{for all} \quad s \geq t$.

Lemma 2

Assume the optimiser visits each state infinitely often. Then,

$$\|\widehat{v}_t - v^*\| \to 0 \text{ as } t \to \infty,$$

and consequently $\widehat{\pi}_t(\cdot) \to \pi^*(\cdot)$.

Theorem 1

For both the modified versions of the *standard* and *indexed* optimisers, when applied to case studies 1 and 2 respectively,

$$\|\widehat{v}_t - v^*\| \to 0 \quad \text{and} \quad \widehat{\pi}_t(\cdot) \to \pi^*(\cdot) \text{ as } t \to \infty.$$

Proof of Lemma 1

We recall from Section 2.2 that for each time $t = 0, 1, \ldots,$

$$\widehat{v}_{t+1}(i,u) = \begin{cases} c_i(u) + \alpha \sum_{j \in S} p_{ij}(u)\widehat{v}_t(j) & \text{if } i = i_t, \\ \widehat{v}_t(i,u) & \text{if } i \neq i_t, \end{cases} \tag{5.2.1}$$

$$\widehat{v}_{t+1}(i) = \min_{u \in U(i)} \{\widehat{v}_{t+1}(i,u)\}, \tag{5.2.2}$$

$$\widehat{\pi}_{t+1}(i) = \arg\min_{u \in U(i)} \{\widehat{v}_{t+1}(i,u)\}. \tag{5.2.3}$$

Let $\pi^*(i)$ be the action for state $i \in S$ under the optimal stationary policy $\pi^*(\cdot)$, so the optimal value function satisfies,

$$v^*(i) = c_i(\pi^*(i)) + \alpha \sum_{j \in S} p_{ij}(\pi^*(i))v^*(j) \quad \text{for all } i \in S. \tag{5.2.4}$$

Let i_t be the current state at time t. Since $v^*(\cdot)$ is the value function that is minimised using the optimal policy $\pi^*(\cdot)$,

$$v^*(i_t) \leq c_{i_t}(u) + \alpha \sum_{j \in S} p_{i_t j}(u)v^*(j) \quad \text{for all } u \in U(i_t). \tag{5.2.5}$$

In addition, since the current value function estimate $\widehat{v}_{t+1}(\cdot)$ at time t is updated using the current greedy policy $\widehat{\pi}_{t+1}(\cdot)$,

$$\widehat{v}_{t+1}(i_t) = c_{i_t}(\widehat{\pi}_{t+1}(i_t)) + \alpha \sum_{j \in S} p_{i_t j}(\widehat{\pi}_{t+1}(i_t))\widehat{v}_t(j)$$

$$\leq c_{i_t}(u) + \alpha \sum_{j \in S} p_{i_t j}(u)\widehat{v}_t(j) \quad \text{for all } u \in U(i_t). \tag{5.2.6}$$

Now using a proof similar to that on Page 34 of Ross (1983),

$$\widehat{v}_{t+1}(i_t) - v^*(i_t) = c_{i_t}(\widehat{\pi}_{t+1}(i_t)) + \alpha \sum_{j \in S} p_{i_t j}(\widehat{\pi}_{t+1}(i_t))\widehat{v}_t(j)$$

$$- c_{i_t}(\pi^*(i_t)) - \alpha \sum_{j \in S} p_{i_t j}(\pi^*(i_t))v^*(j),$$

$$\leq \quad c_{i_t}(\pi^*(i_t)) + \alpha \sum_{j \in S} p_{i_t j}(\pi^*(i_t))\widehat{v}_t(j)$$

$$- \quad c_{i_t}(\pi^*(i_t)) - \alpha \sum_{j \in S} p_{i_t j}(\pi^*(i_t))v^*(j) \quad \text{using Equation (5.2.6),}$$

$$= \quad \alpha \sum_{j \in S} p_{i_t j}(\pi^*(i_t))[\widehat{v}_t(j) - v^*(j)],$$

$$\leq \quad \alpha \sum_{j \in S} p_{i_t j}(\pi^*(i_t)) \sup_j |\widehat{v}_t(j) - v^*(j)|,$$

$$= \quad \alpha \sup_j |\widehat{v}_t(j) - v^*(j)|. \tag{5.2.7}$$

Similarly using Equation (5.2.5) and reversing the roles of $\widehat{v}_{t+1}(i_t)$ and $v^*(i_t)$ for fixed i_t,

$$v^*(i_t) - \widehat{v}_{t+1}(i_t) \quad \leq \quad \alpha \sup_j |v^*(j) - \widehat{v}_t(j)|. \tag{5.2.8}$$

Thus combining Equations (5.2.7) and (5.2.8),

$$\begin{aligned} |\widehat{v}_{t+1}(i_t) - v^*(i_t)| \quad &\leq \quad \alpha \sup_j |\widehat{v}_t(j) - v^*(j)|, \\ &= \quad \alpha \|\widehat{v}_t - v^*\|, \end{aligned} \tag{5.2.9}$$

and using Equations (5.2.1) and (5.2.2),

$$|\widehat{v}_{t+1}(j) - v^*(j)| \quad = \quad |\widehat{v}_t(j) - v^*(j)| \quad \text{for all} \quad j \neq i_t. \tag{5.2.10}$$

Further, from Equations (5.2.9) and (5.2.10) it follows that,

$$\|\widehat{v}_{t+1} - v^*\| \quad \leq \quad \|\widehat{v}_t - v^*\|,$$

and by induction that,

$$\|\widehat{v}_s - v^*\| \quad \leq \quad \|\widehat{v}_t - v^*\| \quad \text{for all} \quad s \geq t. \tag{5.2.11}$$

□

Proof of Lemma 2

Let us define a cycle to be a sequence of updates on the current value function and policy estimates that includes every state $i \in S$ at least once and the final state exactly once. Number the cycles $m = 0, 1, 2, \ldots$. The first cycle is completed at the first instant (after $t = 0$) when we have updated at least once in each state. Let the random time Γ_m be the start of cycle m. We set $\Gamma_0 = 0$.

At each time point t the optimiser visits a state i_t and updates the current value function estimate for that state. For each t let D_t denote the set $\arg\max_{i \in S} |\widehat{v}_t(i) - v^*(i)|$, consisting of all those states $j \in S$ for which $|\widehat{v}_t(j) - v^*(j)| = \|\widehat{v}_t - v^*\|$.

Now let τ be the first time point after Γ_m at which the corresponding set D_t contains a state j_τ which has already been visited at some time point after $\Gamma_m - 1$ but before τ. Furthermore, let ν denote the last time before τ at which j_τ was visited and the current value function estimate for j_τ updated (so $\widehat{v}_{\nu+1}(j_\tau) = \widehat{v}_{\nu+2}(j_\tau) = \ldots = \widehat{v}_\tau(j_\tau)$).

Since all the states are visited at some time in the cycle beginning at Γ_m and ending at $\Gamma_{m+1} - 1$, all states in $D_{\Gamma_{m+1}}$ must have been visited after $\Gamma_m - 1$ and before Γ_{m+1}, so we must have $\tau \leq \Gamma_{m+1}$. Then,

$$\begin{aligned}
\|\widehat{v}_{\Gamma_{m+1}} - v^*\| &\leq \|\widehat{v}_\tau - v^*\| && \text{(from Equation (5.2.11) as } \tau \leq \Gamma_{m+1}\text{)}, \\
&= |\widehat{v}_\tau(j_\tau) - v^*(j_\tau)| && \text{(by definition of } j_\tau\text{)}, \\
&= |\widehat{v}_{\nu+1}(j_\tau) - v^*(j_\tau)|, && \text{(from above)}, \\
&\leq \alpha\|\widehat{v}_\nu - v^*\| && \text{(from Equation (5.2.9) as } j_\tau \text{ is updated at } \nu\text{)}, \\
&\leq \alpha\|\widehat{v}_{\Gamma_m} - v^*\| && \text{(as } \nu \geq \Gamma_m\text{)},
\end{aligned} \tag{5.2.12}$$

and by induction,

$$\|\widehat{v}_{\Gamma_{m+1}} - v^*\| \leq \alpha^m \|\widehat{v}_0 - v^*\|,$$

so $\|\widehat{v}_{\Gamma_m} - v^*\| \to 0$ as $m \to 0$, as $\alpha < 1$. Since $\|\widehat{v}_t - v^*\|$ is non increasing in t from Equation (5.2.11) we can extend the result to give $\|\widehat{v}_t - v^*\| \to 0$ as $t \to \infty$.

Finally since $\widehat{v}_t(\cdot) \to v^*(\cdot)$ as $t \to \infty$, we can apply the theory of Section 1.2.5 which states that any greedy policy with respect to the optimal value function is an optimal policy. Thus $\widehat{\pi}_t(\cdot) \to \pi^*(\cdot)$ as $t \to \infty$. □

Proof of Theorem 1

To prove Theorem 1 we have to show clearly that the modified version of *standard* and *indexed* optimisers visit each state infinitely often in their respective case studies.

In case study 1, even though the *standard* optimiser chooses actions to determine the states it goes to, all of the $p_{ij}(u)$ for all $i, j \in S$ and $u \in U(i)$ are positive. Therefore if the *standard* optimiser was run over an infinite sequence of time, each state would be sampled an infinite number of times.

In case study 2, using the *indexed* optimiser, the probability of moving to a new state is positive unless it has recently been updated. Therefore if the stopping criterion was omitted, each state would be visited infinitely often. □

Bertsekas and Tsitsiklis (1996) give an alternative proof of Lemma 2, restricted to the case of monotonic convergence. Our proof, on the other hand, may be adaptable to study convergence of the current value function estimates when true state transition probabilities are unknown, since in this case the current value function estimates do not change monotonically.

5.3 The Indexed Method in the P-Learner

5.3.1 Introduction

This section discusses the possibility of a passive p-learner motivated by the results presented in Chapter 4. In Chapter 4 we saw that the p-learner experienced difficulties observing the higher valued states, caused by its underlying deficiency when faced with sparsely connected systems. At the time we dealt with the problem by developing an active p-learner (the coupled version) which chose both states and actions, where the index method in the optimiser chose the states in the p-learner at each time step. However, as discussed in Section 4.13, this active case has limited applicability since in many instances we can only observe the state of the (real) system. In this case the only way we can influence the states we go to is by taking actions. We propose a novel index method in the p-learner for future work which observes states using index numbers, independent of the ones defined in the optimiser. These index numbers indirectly force the p-learner to sample each state-action pair ($i \in S, u \in U(i)$). Collectively they are a simple guide to help the p-learner sample states that are observed least often. They balance the need to learn against the wish to use the quickest current route from one state to another.

We first propose a way of defining the indices and then ways of choosing actions.

5.3.2 Defining Indices

Each index $g(i, u, j)$ is defined as a measure of dispreference for taking action u when in state i in order to get to state j. Indices are only updated corresponding the states $j \in S$ we have recently visited. For each state j we look back at the (i, u) combinations we have visited since we were last in state j, or all the way back to beginning of the run if state j has not been visited before. We denote

these as paths. The (i, u) combinations are known as contributions. However, if identical (i, u) combinations exist in a single path, only the most recent one in that path is recorded. The indices themselves are made up of the total time taken to get from (i, u) to j, and are then averaged over the total number of contributions to that point in the process. Recent history tells us by taking action $u \in U(i)$ with the smallest index, the quicker it may lead us to our target state $j \in S$. A weight $0 < \lambda < 1$ could also be incorporated in to the definition of an index, so as to add up all the contributions in the recent history; an idea similar to that of temporal *difference learning.*

5.3.3 Choosing States and Actions

Let $N(j)$ denote the number of visits to state $j \in S$. To use the indices we must choose an action $u \in U(i)$ and a target state $j \in S$. Normally we would choose the state j that minimises $N(j)$ over all $j \in S$ and we only randomise the choice of states if there is a tie. However, we could introduce a threshold L such that if $N(j) > L$ for all $j \in S$, we use the Boltzmann method to choose actions instead.

Chapter 6

Conclusion

This chapter discusses the chief results of the thesis.

In this thesis we addressed the problem of adaptive control in complex stochastic systems when the system parameters were both known and unknown. In Chapter 2 we presented versions of two new algorithms, namely the optimiser and the p-learner. The optimiser is a simulation based method for finding optimal values and optimal policies when the system parameters are known. The p-learner, on the other hand, is used in conjunction with the optimiser when the system parameters are unknown. These methods were applied to typical types of problem as presented in Chapters 3 and 4. The convergence of the solution generated by the optimiser was also considered in Chapter 5.

In the various case studies considered the two most important questions we wanted to address were: (1) relative to the accuracy obtained, how much slower or faster was the optimiser compared to standard DP methods? and (2) what was the computational cost of not knowing the true state transition probabilities? Other important points regarding the learning algorithms were reported as we proceeded. Throughout this chapter we refer to the different versions of algorithms which are clearly labelled in Chapter 4.

Chapter 3 considered a "fully connected" system. This is when it is possible

to move, in a single time step, from a state $i \in S$ to state $j \in S$ given any action $u \in U(i)$. The problem consisted of ten states labelled from 0 to 9 and allowed three actions in each state, labelled from 0 to 2. The optimal Q-values for each state-action pair were plotted in order to identify the optimal action to take in each state. The Q-values were generated using the Pre-Jacobi method. It was seen that even when given perfect information about the system it was not a clear cut decision which action was best in state 8. We colloquially refer to state 8 as the "very difficult state", states 4 and 5 as the "hard states" and the other states as the "easy states". Method 3 was used to choose actions throughout Chapter 3 as it was demonstrated to be a better learning algorithm than Methods 1 and 2. We recall in Method 3 that μ denotes the amount of learning we are willing to do, σ denotes how quickly we move from uniform sampling to focused sampling and Δ denotes the rate at which we discriminate between action $u \in U(i)$ in state $i \in S$ after time μ. Note that the bigger the value of σ, the slower the transition from uniform sampling to focused sampling, and the bigger value of Δ, the faster the rate of discrimination between actions after time μ.

When we applied the optimiser on its own to the "fully connected" case, it was found that the *standard* optimiser produced the same results as the conventional Pre-Jacobi method; the solutions for both methods converged to their corresponding true values. We recall that the *standard* optimiser is a stochastic method and therefore its current value function and policy estimates were updated at only one state per iteration, whereas the Pre-Jacobi updates the current estimates at every state per iteration. In real time the solutions generated by the Pre-Jacobi method converged to the optimal solutions far sooner than those generated by the *standard* optimiser, as the Pre-Jacobi method works $|S|$ times harder at each iteration. However, relatively speaking it was found that the *standard* optimiser was just as fast at finding the optimal solutions as the Pre-Jacobi method.

When the optimiser was run concurrently with the p-learner it was found that

the learning parameter sets for $\{\mu, \sigma, \Delta\}$, used in the p-learner, had a profound affect on the algorithm's ability to perform. This was because of the stochasticity arising from the way we chose actions. This became apparent when we looked at the various convergence criteria discussed in Section 3.5.3.3.

The first of the convergence criteria considered the convergence of the current policy estimate. By looking at the proportion of time spent visiting actions in each state in the p-learner, it was discovered that if the algorithm discriminated too soon (small μ) the p-learner occasionally biased on taking a suboptimal action in state 8 over the optimal one. We then looked at the time intervals at which the current policy estimate converged to the optimal policy; the points at which the current policy estimate settled down to the true solution. These points varied tremendously for different seeds, but this was not the case when the true state transition probabilities were known. The action probabilities for each state for different learning parameter sets were then plotted and it was found that the time intervals at which the current policy estimate oscillated between a suboptimal policy and the optimal one followed a similar pattern to the action probabilities (plotted after time μ in state 8) in the p-learner. In addition to this, it became evident that in the "easy states" any value of σ would have worked equally as well, as the p-learner had done all the work it needed to before the discrimination stage. It was also illustrated that the parameter Δ was most effective in the "hard states". However, it is important to set the parameter Δ to a small value as if the algorithm has not done enough learning by time μ, the algorithm may have a good chance to recover.

The second and third convergence criteria considered the convergence of the current state transition probability and value function estimates respectively. The first criterion used was to look at the quality of the state transition probability estimates as compared to the quality of the true optimal value function estimates. The graphs showed that the computational costs for not knowing the true state transition probabilities were initially fairly high, but the current value

function estimate eventually converged to the true optimal value function estimate. Although this convergence eventually occurred the final error was still far from the true solution. Also, when we compared the performance of the error in the value function estimates with respect to time t, it was found that in the cases where the true state transition probabilities were known the error converged to zero (the Pre-Jacobi being the more dominant method). However, in the case where true state transition probabilities were unknown, the error converged to the wrong value. By plotting the end points for 20 different runs for each learning parameter set, it was discovered that there was a noticeable change in the quality of learning about the model but not about the optimal value function. When the average bias of the end points of the current value function estimates was plotted against the average standard deviation the graphs showed state 8 to be the principal feature. This was because in this particular state the current action estimate sometimes converged to the wrong action. This result was less apparent when we looked at the end points alone.

Chapter 4 considered a "sparsely connected"system known as the machine replacement problem. This is when it is not possible to move, in a single time step, from state $i \in S$ to each state $j \in S$ given any action $u \in U(i)$. In this problem machines deteriorate over time. The lower the value of the state the better their condition. The problem consisted of twelve states labelled from 0 to 11 and allowed two actions in each state (except state 11), labelled 0 and 1. Action 0 denoted the action "to replace" and action 1 denoted the action "do not replace". Only action 0 is available in state 11, as in this state the machine is in its worst conceivable condition and it must be replaced. When we plotted the optimal Q-values using the solutions generated by the Pre-Jacobi method, we saw that the policy decomposed the state space under the stationary policy into two distinct classes: a recurrent class and a transient class. The recurrent class being states $\{0, 1, 2, 3, 4\}$ and the transient class being states $\{5, 6, 7, 8, 9, 10, 11\}$.

When the *standard* optimiser was applied to this sparsely connected system,

it was discovered that the method had trouble finding good estimates of the solutions in states 9, 10 and 11 (again Method 3 was used to choose actions). After further investigation it was found that the reason it had difficulty obtaining good estimates in these states was because the method rarely visited the higher valued states. The same was true in the case when the true state transition probabilities were unknown. These results motivated the development of the *indexed* optimiser.

The *indexed* method ranks states based on the value of their index. These indices encourage the optimiser to visit the states that need updating.

When we applied the *indexed* optimiser to this "sparsely connected" system the problems experienced above in the *standard* version did not arise. Both the *indexed* optimiser and the Pre-Jacobi methods were run using a stopping criteria. If we take into consideration that the *indexed* optimiser updated the current estimates sequentially, it on average worked comparably to the Pre-Jacobi method. However, in the unknown case problems still remained in the p-learner.

When the *indexed* optimiser was run concurrently with the p-learner, the results were analysed using the first, second and third convergence criteria. Using the first convergence criteria it was discovered that the optimiser sampled each state fairly often, but the p-learner did not. This was because we controlled what happened in the optimiser but not in the p-learner. Consequently poor estimates of the important state transition probabilities were obtained. Some of the results we obtained were very misleading, as on many occasions the current policy estimate converged to the true optimal policy even though the state transition probability estimates were subject to large errors. When we looked at the time intervals at which the current policy estimate settled down to the true solution and the time points at which the computation stopped, the intervals varied tremendously for different seeds. This time it was not solely owing to the stochasticity arising from the way we choose actions in the p-learner, but also because the p-learner experienced difficulties observing the higher valued

states irrespective of the learning parameter sets used. The second and third convergence criteria showed that the current value function estimate converged to something that was far from the true solution. When we compared the performance of the three methods simultaneously, again the computational cost of not knowing the true state transition probabilities was fairly high (the Pre-Jacobi method being the more dominant). By plotting the end points for 20 different runs for each learning parameter set, it was discovered that the observations were tightly clustered irrespective of the learning parameter sets used. The average bias of the end points of the current value function estimate when plotted against the average standard deviation demonstrates the consequence of not observing the higher valued states; the higher valued states had higher biases and standard deviations compared with the lower valued states. These results motivated the development of the *coupled* version.

The *coupled* version involved the coupling of the *indexed* optimiser and p-learner. This method uses an active, instead of passive, p-learner. Active p-learners choose both states and actions to enter into the "black box" instead of only choosing actions. Our *coupled* version employs the *indexed* optimiser to choose states in the p-learner, so that the p-learner can sample the higher valued states. Although not widely applicable, the *coupled* version gave us an interesting insight into the bounds of behaviour of the system given that sampling problems in the p-learner could be eradicated. It was discovered that by plotting the end points of the different runs for different learning parameter sets the algorithms performances were far better than in the "fully connected" problem. However, in this case the p-learner did not have so many state transition probabilities to learn about. Also, when we plotted the average bias of the current value function estimates against the average standard deviation the results obtained were good, but not as good as those obtained in the known case.

To link Chapters 3 and 4 we analysed the performance of the *standard* optimiser with that of the *indexed* optimiser on the "fully connected" system. It was

found that the the *indexed* optimiser out performed the *standard* optimiser when the system parameters were known, and was at least as good when they were unknown.

Chapter 5 considers the convergence of the optimiser of the solution generated by the optimiser. It was proved that as long as each state is visited infinitely often in the optimiser the current estimates will converge to their corresponding true solutions. In practice, case studies 1 and 2 showed that if all of the states are visited fairly often, in the known case, the current estimates converge to the corresponding true values in a short period of time. Proving the convergence of estimates in the unknown case is left for future work, but it is still not clear that our algorithms solutions will converge to the true solutions over an infinite time period. It was proposed that the p-learner might be modified so that the algorithm might track the process over time, rather than relying on knowing the structure of the problem, and thus behave similarly to the *indexed* optimiser. This would present us with a universally applicable algorithm for instances in which the true state transition probabilities are unknown.

In conclusion, the Pre-Jacobi method is computationally expensive, thus it may be more efficient to use a version of our optimiser instead. We have demonstrated that, in relative terms, the optimiser is just as fast in finding the true solutions of a problem as the Pre-Jacobi method, as well as being more universally applicable. It is also clear that the computational cost of running the optimiser concurrently with the p-learner when we know the true state transition probabilities is high. Though this is the case, running it enables us to tackle problems in which we could not normally use full DP. The various case studies have shown that our methods can be adapted to problems with a particular type of structure in complex stochastic systems.

Appendix A

The following state transition matrices were used in the "fully connected" problem described in Chapter 3. They are defined for each state labelled from 0 to 9 and each action labelled from 0 to 2. The first, second and third transition matrices listed below denote the state transition matrices corresonding to actions 0, 1 and 2 respectively. The state transition probabilities were generated using a random number generator. Each row was normalised so that the sum of each row added to 1.

$$\begin{pmatrix}
0.0955 & 0.1167 & 0.0858 & 0.1408 & 0.0530 & 0.0340 & 0.0318 & 0.2347 & 0.1198 & 0.0879 \\
0.0817 & 0.0841 & 0.1602 & 0.0086 & 0.1736 & 0.0791 & 0.1696 & 0.0401 & 0.1081 & 0.0950 \\
0.1677 & 0.0905 & 0.0272 & 0.1315 & 0.1117 & 0.0359 & 0.0974 & 0.0909 & 0.0921 & 0.1551 \\
0.0283 & 0.0256 & 0.1910 & 0.0707 & 0.2267 & 0.1646 & 0.0787 & 0.0445 & 0.1538 & 0.0161 \\
0.0562 & 0.1526 & 0.0291 & 0.0409 & 0.1339 & 0.1345 & 0.0116 & 0.2082 & 0.1486 & 0.0844 \\
0.0145 & 0.0726 & 0.0450 & 0.1754 & 0.1769 & 0.0951 & 0.0063 & 0.1382 & 0.1823 & 0.0939 \\
0.0251 & 0.0960 & 0.1517 & 0.0756 & 0.0746 & 0.1583 & 0.0004 & 0.0513 & 0.1852 & 0.1818 \\
0.0029 & 0.0834 & 0.0412 & 0.0757 & 0.0562 & 0.1370 & 0.0860 & 0.2071 & 0.1382 & 0.1723 \\
0.1161 & 0.0484 & 0.1134 & 0.1186 & 0.0442 & 0.1171 & 0.1944 & 0.1906 & 0.0144 & 0.0430 \\
0.1096 & 0.0130 & 0.1425 & 0.0634 & 0.0516 & 0.1065 & 0.1662 & 0.0950 & 0.1659 & 0.0864
\end{pmatrix}$$

$$\begin{pmatrix}
0.0889 & 0.0274 & 0.0272 & 0.1988 & 0.2092 & 0.1390 & 0.0075 & 0.0104 & 0.0520 & 0.2396 \\
0.2184 & 0.0028 & 0.1763 & 0.0059 & 0.0629 & 0.0023 & 0.1679 & 0.2443 & 0.1029 & 0.0163 \\
0.1636 & 0.0354 & 0.0740 & 0.1153 & 0.0993 & 0.1383 & 0.0009 & 0.1481 & 0.1873 & 0.0378 \\
0.0154 & 0.1997 & 0.1371 & 0.1074 & 0.2682 & 0.0042 & 0.0811 & 0.0577 & 0.0207 & 0.1084 \\
0.0003 & 0.0263 & 0.1877 & 0.1364 & 0.1450 & 0.0340 & 0.1658 & 0.0602 & 0.0539 & 0.1902 \\
0.0004 & 0.1717 & 0.0980 & 0.1657 & 0.0774 & 0.0234 & 0.1639 & 0.0762 & 0.0456 & 0.1778 \\
0.0799 & 0.0589 & 0.0300 & 0.0869 & 0.1024 & 0.0968 & 0.1083 & 0.1799 & 0.2406 & 0.0164 \\
0.1280 & 0.0810 & 0.0897 & 0.0655 & 0.0937 & 0.1575 & 0.0619 & 0.0149 & 0.1020 & 0.2059 \\
0.0637 & 0.0881 & 0.1201 & 0.0814 & 0.1387 & 0.1242 & 0.1622 & 0.1448 & 0.0070 & 0.0699 \\
0.1004 & 0.1434 & 0.0588 & 0.1371 & 0.1542 & 0.0996 & 0.1226 & 0.0016 & 0.0102 & 0.1722
\end{pmatrix}$$

$$\begin{pmatrix}
0.1269 & 0.1500 & 0.1637 & 0.1077 & 0.0161 & 0.0909 & 0.1047 & 0.1121 & 0.0335 & 0.0944 \\
0.0723 & 0.0836 & 0.0411 & 0.1622 & 0.0723 & 0.0896 & 0.1588 & 0.0973 & 0.0847 & 0.1383 \\
0.1170 & 0.1245 & 0.0577 & 0.1461 & 0.0486 & 0.1784 & 0.1225 & 0.0101 & 0.1331 & 0.0620 \\
0.0289 & 0.1054 & 0.1455 & 0.0375 & 0.0907 & 0.1059 & 0.0977 & 0.1295 & 0.1100 & 0.1489 \\
0.1679 & 0.0954 & 0.0286 & 0.0269 & 0.1549 & 0.1582 & 0.0473 & 0.0142 & 0.1507 & 0.1558 \\
0.0431 & 0.1839 & 0.0779 & 0.1213 & 0.1168 & 0.1818 & 0.1091 & 0.1326 & 0.0319 & 0.0017 \\
0.0135 & 0.0063 & 0.0495 & 0.0301 & 0.0554 & 0.0554 & 0.1652 & 0.1010 & 0.2172 & 0.3064 \\
0.1528 & 0.0144 & 0.1705 & 0.1522 & 0.0137 & 0.0602 & 0.0795 & 0.2062 & 0.0750 & 0.0754 \\
0.0863 & 0.1298 & 0.0834 & 0.0235 & 0.1607 & 0.1192 & 0.1033 & 0.1934 & 0.0956 & 0.0048 \\
0.0054 & 0.0390 & 0.0002 & 0.0681 & 0.2530 & 0.0788 & 0.1233 & 0.0269 & 0.1385 & 0.2668
\end{pmatrix}$$

References

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning To Act Using Real-Time Dynamic Programming. *Artificial Intelligence*, **72**, 81–138.

Barto, A. G. and Singh, S. P. (1990). On the computational economics of reinforcement learning. In M. Kaufmann (Ed.), *Proceedings of the 1990 Connectionist Models Summer School*, San Mateo, California.

Bellman, R. (1957). *Dynamic Programming*. New Jersey: Princeton University Press.

Bertsekas, D. P. (1982). Distributed Dynamic Programming. *IEEE Transactions on Automatic Control*, **27**, 610–616.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, New Jersey: Prentice-Hall.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Belmont, Massachusetts: Athena Scientific.

Blackwell, D. (1965). Discounted Dynamic Programming. *Ann Math Statist.*, **36**, 226–235.

Dayan, P. (1992). The convergence of $TD(\lambda)$ for general λ. *Machine Learning*, **8**, 341–362.

Dayan, P. and Sejnowski, T. J. (1994). $TD(\lambda)$ converges with probability 1. *Machine Learning*, **14**, 295–301.

References.

Denardo, E. V. (1982). *Dynamic Programming: Models and Applications.* Englewood Cliffs, New Jersey.

Fiechter, C. N. (1994). Efficient Reinforcement Learning. In A. Press (Ed.), *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pp. 88–97.

Gittins, J. C. (1989). *Multi-armed Bandit Allocation Indices.* Wiley-Interscience in systems and optimization. Chichester, New York: Wiley.

Humphrys, M. (1996). *Action Selection Methods Using Reinforcement Learning.* Ph.d thesis, Cambridge University, England.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On The Covergence of Stochastic Iterative Dynamic Programming. *Neural Computation*, **6**, 1185–1201.

John, G. H. (1995). When the best move isn't optimal. Technical report, Unpublished manusript, available through URL ftp://starry.stanford.edu/pub/gjohn/papers/rein-nips.ps.

Jones, R. A. and Collins, E. J. (1998). Solving Discounted Markov Decision Problems Using Neuro-Dynamic Programming. In *COMPSTAT 1998, Proceedings in Computational Statistics*, Volume 13, IACR-Rothamsted, pp. 53–54.

Kaelbling, L. P. (1993). *Learning in Embedded Systems.* Cambridge, Massachusetts: MIT Press.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence*, 237–285.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimisation by Simulated Annealing. *Science*, **220**, 671–680.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized Sweeping: Reinforcement Learning with less data and less real time. *Machine Learning*, **13**.

References.

O'Hagan, A. (1994). *Kendall's Advanced Theory of Statistics: Bayesian Inference.* London: E. Arnold.

Peng, J. and Williams, R. J. (1993). Efficient Learning and Planning within The Dyna Framework. *Adaptive Behaviour*, **1**, 437–454.

Peng, J. and Williams, R. J. (1994). Incremental Multi-step Q-Learning. In M. Kaufmann (Ed.), *Proceedings of the Eleventh International Conference on Machine Learning*, San Franscisco, California, pp. 226–232.

Porteus, E. L. (1975). Bounds and Transformations for Discounted Finite Markov Decision Chains. *Operations Research*, **23**, 761–764.

Puterman, M. L. (1994). *Markov Decision Process: Discrete Stochastic Dynamic Programming.* New York: John Wiley and Sons.

Robbins, H. and Monro, S. (1951). A stochastic approximation model. *Annals of mathematical statistics*, **22**, 400–407.

Ross, S. (1983). *Introduction to Stochastic Dynamic Programming.* New York: Academic Press.

Sato, M., Abe, K., and Takeda, H. (1988). Learning Control of Finite Markov Chains with Explicit Trade-off Between Estimation and Control. *IEEE Transactions on Systems, Man, and Cybernetics*, **18**, 677–684.

Singh, S. P. (1994). *Learning To Solve Markovian Decision Processes.* Ph.d thesis, University of Massachusetts, Amherst, Massachusetts.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement Learning with Replacing Eligibility Trees. *Machine Learning*, **22**, 123–158.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* Ph.d thesis, University of Massachusetts, Amherst, Massachusetts.

Sutton, R. S. (1988). Learning to Predict By The Methods of Temporal Differences. *Machine Learning*, **3**, 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In M. Kaufmann (Ed.), *In Proceedings of the Seventh International Workshop*, San Mateo, California, pp. 216–224.

Sutton, R. S. (1991a). Dyna, an Integrated Architecture for Learning, Planning, and Reacting. In *Working Notes of the 1991 AAAI Spring Symposium*, pp. 151–155.

Sutton, R. S. (1991b). Planning by Incremental Dynamic Programming. In M. Kaufmann (Ed.), *In Proceedings of the Eighth International Workshop on Machine Learning*, pp. 353–357.

Thurn, S. B. (1992). The Role of Exploration in Learning Control. *Handbook of Intelligent Control: Neural Fuzzy and Adaptive Approaches, Van Nostrand Reinhold, New York.*

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-Learning. *Machine Learning*, **16**, 185–202.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* Ph.d thesis, Cambridge University, England.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, **8**, 279–292.

White, D. J. (1963). Dynamic Programming, Markov Chains, and the Method of Successive Approximations. *Journal of Mathematical Analysis and Applications*, **6**, 373–376.